



Mieso K. Denko
Laurence Tianruo Yang
Yan Zhang
Editors

Autonomic Computing and Networking

 Springer

Autonomic Computing and Networking

Mieso K. Denko · Laurence Tianruo Yang ·
Yan Zhang
Editors

Autonomic Computing and Networking

 Springer

Editors

Mieso K. Denko
Department of Computing and
Information Science
University of Guelph
Reynolds Building
Guelph, ON N1G 2W1
Canada
denko@cis.uoguelph.ca

Laurence Tianruo Yang
Department of Computer Science
Francis Xavier University
Antigonish, NS B2G 2W5
Canada
ltyang@gmail.com

Yan Zhang
Simula Research Laboratory
Norway
yanzhang@ieee.org

ISBN 978-0-387-89827-8 e-ISBN 978-0-387-89828-5
DOI 10.1007/978-0-387-89828-5

Library of Congress Control Number: 2008940265

© Springer Science+Business Media, LLC 2009

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

springer.com

Preface

Autonomic computing and networking are emerging paradigms that allow for the creation of self-managing and self-controlling environments by employing distributed algorithms and context-awareness to dynamically control networking functions without human interventions. Autonomic networking is characterized by recovery from failures and malfunctions and agility to changing networking environments and self-optimization. The self-control and management features can help overcome the growing complexity and heterogeneity of existing communication networks and systems. The realization of fully autonomic heterogeneous networking requires fundamental research challenges in all aspects of computing, networking, communications, and other related fields.

This book, with chapters contributed by prominent researchers from academia and industry, will serve as a technical guide and reference material for engineers, scientists, practitioners, and researchers by providing them with state-of-the-art research findings and future opportunities and trends. These contributions include state-of-the-art architectures, protocols, technologies, and applications in pervasive computing and wireless networking. In particular, the book covers existing and emerging communications and computing models, design architectures, mobile and wireless applications, technologies, and research issues in autonomic computing systems and communications.

The book has 18 chapters organized into two sections: autonomic computing and autonomic networking. Each section contains nine chapters addressing existing and emerging architectures, protocols, and applications.

Part I Autonomic Computing

This section consists of Chapters 1–9 and covers various topics on autonomic computing systems and applications. Chapter 1 by Radu discusses a generic autonomic computing framework for the development of self-managing systems. A prototype implementation of the reconfigurable policy engine is used to develop autonomic solutions in case studies from several application domains.

Chapter 2 by Garlan et al. presents a system called Rainbow that uses software architecture models and styles to support self-adaptation. The framework provides

general and reusable infrastructures with well-defined customization points, allowing engineers to systematically customize Rainbow for particular systems. Chapter 3 by Mpitzopoulos et al. discusses mobile agent-based middleware solutions for autonomic data fusion tasks. Chapter 4 by Hagimont et al. presents a component-based autonomic management system for legacy software. It describes the design and implementation of such a system and evaluates different uses. Chapter 5 by Brock and Goscinski proposes a dynamic web services description language for supporting autonomic computing. The framework allows the attributes of web services to be visible, thus allowing the autonomic system to better cater to the installation and use of new components. Chapter 6 by Oliveri et al. discusses a bio-inspired cognitive radio for dynamic spectrum access. Autonomic bio-inspired approaches and spectral access are also discussed. Chapter 7 by Boucadair discusses the introduction of autonomous behaviors to IP multimedia subsystem (IMS)-based architectures. Solutions covered aim at enhancing the robustness and the availability of current IMS-based architectures owing to the activation of autonomic-like techniques. Chapter 8 by Bixio et al. discusses the cognition-based distributed spectrum sensing for autonomic wireless systems. Finally, in Chapter 9, Kwok presents an autonomic peer-to-peer systems with a focus on incentive and security issues.

Part II: Autonomic Networking

This section consists of Chapters 10–18 with a focus on autonomic networking and communications.

Chapter 10 by Boutaba et al. discusses autonomic networks with focus on knowledge management and self-stabilization. In-depth discussions of basic concepts, research challenges, and their importance for the success of autonomic networks are presented. Chapter 11 by Yu et al. discusses autonomic wireless sensor networks. The chapter has an in-depth discussion of existing research activities in this area. Chapter 12 by Wada et al. discusses a model-driven development environment for biologically inspired autonomic network applications. The chapter proposes and evaluates a new development environment, called iNetLab, which can improve the productivity of designing, maintaining, and tuning operational policies in autonomic network applications. Chapter 13 by Cascado et al. discusses network reconfiguration in high-performance interconnection networks. Chapter 14 by Zulkernine et al. discusses autonomic management of networked web service-based processes. The authors discuss web services management from service providers' and service consumers' perspectives.

Chapter 15 by Zseby et al. discusses self-protection in autonomic and related networks. Chapter 16 by Cong-Vinh discusses the formal aspects of self-* in autonomic networked computing systems. Chapter 17 by Alouf et al. discusses autonomic information diffusion in intermittently connected networks. The chapter proposes a framework for designing autonomic information diffusion mechanisms using techniques and tools drawn from evolutionary computing research. Finally, Chapter 18

by He et al. presents dynamic and fair spectrum access mechanism for autonomous communications.

This book has the following salient features:

- Provides a comprehensive reference on autonomic computing and networking.
- Presents state-of-the-art techniques in autonomic computing and networking.
- Contains illustrative figures enabling easy reading.
- Discusses emerging trends and open research problems in autonomic computing and networking.

We owe our deepest gratitude to all the authors for their valuable contribution to this book and their great efforts. All of them are extremely professional and cooperative. We wish to express our thanks to Springer especially Katelyn Stanne, Caitlin Womersley, and Jason Ward for their support and guidance during the preparation of this book. A special thank also goes to our families and friends for their constant encouragement, patience, and understanding throughout this project.

The book serves as a comprehensive and essential reference on autonomic computing and networking and is intended as a textbook for senior undergraduate and graduate-level courses. It can also be used as a supplementary textbook for undergraduate courses. The book is a useful resource for the students and researchers to learn autonomic computing and networking. In addition, it will be valuable to professionals from both the academia and industry and generally serves instant appeal to the people who would like to contribute to autonomic computing and networking technologies.

We welcome and appreciate your feedback and hope you enjoy reading the book.

Mieso K. Denko
Ontario, Canada

Laurence T. Yang
Nova Scotia, Canada

Yan Zhang
Oslo, Norway

Contents

Part I Autonomic Computing

General-Purpose Autonomic Computing	3
Radu Calinescu	
Software Architecture-Based Self-Adaptation	31
David Garlan, Bradley Schmerl, and Shang-Wen Cheng	
Mobile Agent Middleware for Autonomic Data Fusion in Wireless Sensor Networks	57
Aristides Mpitzopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou	
Component-Based Autonomic Management for Legacy Software	83
Daniel Hagimont, Patricia Stolf, Laurent Broto, and Noel De Palma	
Dynamic WSDL for Supporting Autonomic Computing	105
Michael Brock and Andrzej Goscinski	
Bio-inspired Cognitive Radio for Dynamic Spectrum Access	131
Giacomo Oliveri, Marina Ottonello, and Carlo S. Regazzoni	
Introducing Autonomous Behaviors into IMS-Based Architectures	155
Mohamed Boucadair	
Embodied Cognition-Based Distributed Spectrum Sensing for Autonomic Wireless Systems	179
Luca Bixio, Andrea F. Cattoni, Carlo S. Regazzoni, and Pramod K. Varshney	
Autonomic Peer-to-Peer Systems: Incentive and Security Issues	205
Yu-Kwong Kwok	

Part II Autonomic Networking

Toward Autonomic Networks: Knowledge Management and Self-Stabilization	239
Raouf Boutaba, Jin Xiao, and Qi Zhang	
Autonomic Networking in Wireless Sensor Networks	261
Mengjie Yu, Hala Mokhtar, and Madjid Merabti	
iNetLab: A Model-Driven Development and Performance Engineering Environment for Autonomic Network Applications	285
Hiroshi Wada, Chonho Lee, Junichi Suzuki, and Tetsuo Otani	
Network Reconfiguration in High-Performance Interconnection Networks	313
R. Casado, A. Bermúdez, A. Robles-Gómez, O. Lysne, T. Skeie, Å.G. Solheim, and T. Sødning	
Autonomic Management of Networked Web Services-Based Processes	333
Farhana H. Zulkernine, Wendy Powley, and Patrick Martin	
Concepts for Self-Protection	355
Tanja Zseby, Heiko Pfeffer, and Stephan Steglich	
Formal Aspects of Self-* in Autonomic Networked Computing Systems . . .	381
Phan Cong-Vinh	
Autonomic Information Diffusion in Intermittently Connected Networks	411
Sara Alouf, Iacopo Carreras, Álvaro Fialho, Daniele Miorandi, and Giovanni Neglia	
Dynamic and Fair Spectrum Access for Autonomous Communications	435
Jianhua He, Jie Xiang, Yan Zhang, and Zuoyin Tang	
Index	455

Contributors

Sara Alouf INRIA, Sophia Antipolis, France, sara.alouf@sophia.inria.fr

A. Bermúdez Universidad de Castilla-La Mancha, I3A Campus Universitario s/n, 02071 Albacete, Spain, abermu@dsi.uclm.es

Luca Bixio Department of Biophysical and Electronic Engineering, University of Genova, Via Opera Pia 11a, 16145 Genova, Italy, luca.bixio@dibe.unige.it

Mohamed Boucadair France Télécom R&D, 42 Rue des coutures, 14066 Caen Cedex, France, mohamed.boucadair@orange-ftgroup.com

Raouf Boutaba David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, rboutaba@cs.uwaterloo.ca

Michael Brock Deakin University, Pigdons Road, Waurn Ponds, Geelong, Victoria 3217, Australia, mrab@deakin.edu.au

Laurent Broto UPS, Toulouse, France, broto@irit.fr

Radu Calinescu Computing Laboratory, University of Oxford, Oxford, England UK, Radu.Calinescu@comlab.ox.ac.uk

Iacopo Carreras CREATE-NET, Trento, Italy, iacopo.carreras@create-net.org

R. Casado Universidad de Castilla-La Mancha, I3A Campus Universitario s/n, 02071 Albacete, Spain, rcasado@dsi.uclm.es

Andrea F. Cattoni Department of Biophysical and Electronic Engineering, University of Genova, Via Opera Pia 11a, 16145 Genova, Italy, cattoni@dibe.unige.it

Shang-Wen Cheng Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA, zensoul@cs.cmu.edu

Phan Cong-Vinh London South Bank University, Borough Road, London SE1 0AA, United Kingdom, phanvc@ieee.org

Noel De Palma INPG, Grenoble, France, depalma@inrialpes.fr

Álvaro Roberto Silvestre Fialho INRIA, Sophia Antipolis, France, Now at Microsoft Research-INRIA Joint Centre, Orsay, France, alvaro.fialho@inria.fr

David Garlan Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA, garlan@cs.cmu.edu

Damianos Gavalas Dept of Cultural Technology and Communication, University of the Aegean Address of Institute, Lesvos, Greece, dgavalas@aegean.gr

Andrzej Goscinski Deakin University, Pigdons Road, Waurn Ponds, Geelong Victoria 3217, Australia, ang@deakin.edu.au

Daniel Hagimont INPT, Toulouse, France, hagimont@enseiht.fr

Jianhua He Institute of Advanced Telecommunications, Swansea University, Swansea SA2 8PP, UK, j.he@swansea.ac.uk

Charalampos Konstantopoulos Research Academic Computer Technology Institute, Patras, Greece, konstant@cti.gr

Yu-Kwong Kwok Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80526-1373, USA, Ricky.Kwok@colostate.edu

Chonho Lee University of Massachusetts, Boston, MA, USA
chonho@cs.umb.edu

O. Lysne University of Oslo, Simula Research Laboratory, P.O. Box 134, N-1325 Lysaker, Norway, olavly@simula.no

Patrick Martin School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada, martin@cs.queensu.ca

Madjid Merabti School of Computing and Mathematical Science, Liverpool John Moores University, Byrom Street, Liverpool, UK, M.Merabti@ljmu.ac.uk

Daniele Miorandi CREATE-NET, Trento, Italy, daniele.miorandi@create-net.org

Hala Mokhtar School of Computing and Mathematical Science, Liverpool John Moores University, Byrom Street, Liverpool, UK, H.M.Mokhtar@ljmu.ac.uk

Aristides Mpitziopoulos Dept of Cultural Technology and Communication, University of the Aegean Address of Institute, Lesvos, Greece, crmaris@aegean.gr

Giovanni Neglia INRIA, Sophia Antipolis, France University of Palermo, P alermo, Italy, giovanni.neglia@ieee.org

Giacomo Oliveri Department of Biophysical and Electronic Engineering, University of Genova, Via Opera Pia 11a, 16145 Genova, Italy,
giacomo.oliveri@dibe.unige.it

Tetsuo Otani Central Research Institute of Electric Power Industry
ohtani@criepi.denken.or.jp

Marina Ottonello Department of Biophysical and Electronic Engineering,
University of Genova, Via Opera Pia 11a, 16145 Genova, Italy,
marina@dibe.unige.it

Grammati Pantziou Department of Informatics, Technological Educational
Institution of Athens, Athens, Greece, pantziou@teiath.gr

Heiko Pfeffer Fraunhofer Institute Fokus, Berlin, Germany
heiko.pfeffer@fokus.fraunhofer.de

Wendy Powley School of Computing, Queen's University, Kingston, ON K7L
3N6, Canada, wendy@cs.queensu.ca

Carlo S. Regazzoni Department of Biophysical and Electronic Engineering,
University of Genova, Via Opera Pia 11a, 16145 Genova, Italy, carlo@dibe.unige.it

A. Robles-Gómez Universidad de Castilla-La Mancha, I3A Campus Universitario
s/n, 02071 Albacete, Spain, arobles@dsi.uclm.es

Bradley Schmerl Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA
15213, USA, schmerl@cs.cmu.edu

T. Skeie University of Oslo, Simula Research Laboratory, P.O. Box 134, N-1325
Lysaker, Norway, tskeie@simula.no

T. Sødning University of Oslo, Simula Research Laboratory, P.O. Box 134,
N-1325 Lysaker, Norway, tsodning@simula.no

A.G. Solheim University of Oslo, Simula Research Laboratory, P.O. Box 134,
N-1325 Lysaker, Norway, aashig@simula.no

Stephan Steglich Fraunhofer Institute Fokus, Berlin, Germany
stephan.steglich@fokus.fraunhofer.de

Patricia Stolf IUFM, Toulouse, France, stolf@irit.fr

Junichi Suzuki University of Massachusetts, Boston, MA jxsq@cs.umb.edu

Zuoyin Tang

Pramod K. Varshney Department of Electrical Engineering and Computer
Science, Syracuse University, NY, USA, varshney@syr.edu

Hiroshi Wada University of Massachusetts, Boston, MA, USA, fshu@cs.umb.edu

Jie Xiang Simula Research Laboratory, Martin Linges vei 17, IT Fornebu,
P.O.Box 134, No-1325 Lysaker, Norway, jxiang@simula.no

Jin Xiao David R. Cheriton School of Computer Science, University of Waterloo,
Waterloo, ON, Canada, j2xiao@cs.uwaterloo.ca

Mengjie Yu School of Computing and Mathematical Science, Liverpool John
Moores University, Byrom Street, Liverpool UK, M.Yu@2001.ljmu.ac.uk

Qi Zhang David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, q8zhangg@cs.uwaterloo.ca

Tanja Zseby Fraunhofer Institute Fokus, Berlin, Germany
tanja.zseby@fokus.fraunhofer.de

Farhana H. Zulkernine School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada, farhana@cs.queensu.ca

Yan Zhang Simula Research Laboratory, Norway, yanzhang@iee.org

Zuoyin Tang Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow G1 1XW, UK, Zuoyin.Tang@strath.ac.uk

Part I
Autonomic Computing

General-Purpose Autonomic Computing

Radu Calinescu

Abstract The success of mainstream computing is largely due to the widespread availability of general-purpose architectures and of generic approaches that can be used to solve real-world problems cost-effectively and across a broad range of application domains. In this chapter, we propose that a similar generic framework is used to make the development of autonomic solutions cost effective, and to establish autonomic computing as a major approach to managing the complexity of today's large-scale systems and systems of systems. To demonstrate the feasibility of *general-purpose autonomic computing*, we introduce a generic autonomic computing framework comprising a policy-based autonomic architecture and a novel four-step method for the effective development of self-managing systems. A prototype implementation of the reconfigurable policy engine at the core of our architecture is then used to develop autonomic solutions for case studies from several application domains. Looking into the future, we describe a methodology for the engineering of self-managing systems that extends and generalises our autonomic computing framework further.

1 Introduction

The last decade has brought revolutionary transformations to the way in which information and communication technologies (ICT) are used to conduct business and research and to provide services in all sectors of the society [26]. The ability to accomplish more, faster and on a broader scale through expert use of ever more complex ICT systems is at the core of today's scientific discoveries, newly emerged services and everyday life. Autonomic computing represents an effective approach to managing the spiralling complexity of these systems by delegating their configuration, optimisation, repair and protection to the systems themselves [15, 21].

R. Calinescu (✉)
Computing Laboratory, University of Oxford, Oxford, England, UK
e-mail: Radu.Calinescu@comlab.ox.ac.uk

The research efforts of the past few years have generated a wealth of knowledge on what autonomic systems should look like [9, 13, 21, 31, 34] and what best practices to follow in building them [4, 16, 41, 43]. This progress is to a great extent a by-product of the effort that went into the development of successful autonomic solutions addressing specific management tasks in real-world applications [8, 25, 27, 40, 42]. While these developments demonstrate the feasibility of the autonomic computing approach to complexity management, the current use of bespoke and domain-specific architectures, and of dedicated models and policies limits significantly the cost-effectiveness and reusability of today’s autonomic solutions.

These limitations resemble the problems encountered in the early days of mainstream computing, and overcome successfully through the use of general-purpose architectures and generic approaches for the development of real-world applications across multiple application domains. We therefore propose that an equally generic framework is used to make the development of self-managing systems cost-effective, and to drive standardisation, component reuse and user adoption in the realm of autonomic computing. Given that policy-based autonomic computing represents the most advanced approach to developing self-managing systems of practical utility, we describe below the criteria that a policy-based autonomic computing framework needs to satisfy in order to qualify as “general purpose”:

- C1 Support for the whole range of software, hardware and data components encountered in real-world ICT systems.** To enable the development of effective autonomic systems for real-world applications, the framework should support the organisation of heterogeneous collections of existing and future ICT components into self-managing systems. Both components specifically designed for inclusion into a self-managing system (i.e., *autonomic-enabled ICT resources*) and components not originally intended for this purpose (i.e., *legacy ICT resources*) should be catered for.¹
- C2 Support for a broad spectrum of self-* functional areas and autonomic computing policies.** The framework should aid the development of self-management capabilities spawning a rich spectrum of self-* functional areas, e.g., self-configuration, self-healing, self-optimisation and self-protection [21, 31, 34]. This must be achieved through supporting all types of autonomic computing policies, including action, goal and utility-function policies [44, 45].
- C3 Support for the cost-effective development of self-managing systems for a large variety of application domains and use cases.** The framework must reduce the effort and costs incurred in the development of today’s autonomic systems significantly through enabling the extensive reuse of components and the sharing of autonomic computing models and policies. It should drive the standardisation of interfaces, policies, models and components for autonomic computing, and should allow and encourage the modular development of complex self-managing systems and systems of systems. Last but not least, the framework must provide a generic method for developing autonomic systems from any combination of legacy and/or autonomic-enabled ICT resources.

¹ The ICT components to be integrated into an autonomic system will be termed (*ICT resources*).

To demonstrate the feasibility of general-purpose autonomic computing, we introduce a novel policy-based autonomic computing framework comprising an autonomic architecture designed around a reconfigurable policy engine, and a four-step method for the effective development of self-managing systems. This framework builds on recent advances in autonomic computing [9, 13, 17, 34], and extends the author’s previous work in this area [4–7] in several new directions. Thus, we describe for the first time how multiple instances of the same general-purpose autonomic architecture can be organised into self-managing systems of systems by means of a new type of autonomic policy termed a *resource-definition policy*. Also, we present the first-ever integration of quantitative model checking techniques [23, 24] into autonomic policy engines, and show how the use of this new capability enables the specification of powerful utility-function policies. Finally, we present a new four-step method for the development of self-managing systems starting from a model of their ICT resources, and we illustrate its application to several case studies that spawn different application domains and employ a wide range of policy types.

The remainder of the chapter is organised as follows. In Sect. 2, we contrast our framework with other approaches to autonomic solution development. We then describe the general-purpose autonomic architecture and the reconfigurable policy engine at its core in Sects. 3 and 4, respectively. A prototype implementation of the policy engine is presented in Sect. 5, followed by the description of our generic method for the development of self-managing systems in Sect. 6, and by several case studies that illustrate its use in a number of different real-world applications in Sect. 7. Section 8 analysis the extent to which our candidate general-purpose autonomic framework satisfies the criteria stated at the beginning of the chapter, and suggests ways for extending our current results.

2 Related Work

The autonomic infrastructure proposed in [35] is retrofitting autonomic functionality onto legacy systems by using *sensors* to collect resource data, *gauges* to interpret these data and *controllers* to decide the “adaptations” to be enforced on the managed systems through *effectors*. This infrastructure was successfully used to monitor, analyse and control legacy systems in applications such as spam detection, instant messaging quality-of-service management and load balancing for geographical information systems [19]. Our framework is building on the powerful approach in [19, 35], and has the added capability to handle heterogeneous types of resources unknown until runtime, and to support the development of autonomic systems of systems through the use of resource-definition policies.

In [20], the authors define an autonomic architecture meta-model that extends IBM’s autonomic computing blueprint [16], and use a model-driven process to partly automate the generation of instances of this meta-model. Each instance is a special-purpose *organic computing system* that can handle the use cases defined by the model used for its generation. Our general-purpose autonomic architecture eliminates the need for the 19-activity generation process described in [20] by using

a universal policy engine that can be dynamically redeployed to handle any use cases encoded within its resource model and policy set.

Several research projects propose the use of model-driven architecture (MDA) techniques to develop autonomic computing policies and self-managing systems starting from high-level behavioural models of the system or of its components [10, 36, 39]. Two of these approaches [10, 36] are targeted at bespoke systems whose components already exhibit sophisticated autonomic behaviour, and thus cannot be readily extended to handle generic legacy resources. In contrast, our framework can accommodate any type of ICT resource whose characteristics can be modelled as described in Sect. 6. The preliminary work described in [39] is closer to our approach in that it advocates the importance of using MDA techniques in the development of generic self-managing systems; however, the authors do not substantiate their proposal with any concrete solution, but rather qualify it as an open challenge.

A number of other projects have investigated isolated aspects related to the development of autonomic systems out of non-autonomic components. Some of these projects addressed the standardisation of the policy information model, with the Policy Core Information Model [30] representing the most prominent outcome of this work. Recent efforts such as Oasis' Web Services Distributed Management (WSDM) project were directed at the standardisation of the interfaces through which the manageability of a resource is made available to other applications [32]. An integrated development environment for the implementation of WSDM-compliant interfaces is currently available from IBM [17].

In [12], the authors take a view similar to ours by introducing a paradigm termed *model-driven autonomic computing*, and explaining that the model-based validation of self-management decisions represents a more reliable and flexible approach than the use of pre-set policies. A powerful hierarchical model of NASA's Autonomous Nano-Technology Swarm missions is successfully used in [12] to achieve the self-managing functionality that these missions depend on, and thus to illustrate the benefits of the approach. Our work complements the results in [12] with a new model-based approach to developing self-management functionality and a generic method that uses existing tools and standards for the implementation of autonomic systems.

Finally, we build on recent advances in component-based programming, by using an approach to ICT resource composition and dynamic configuration that resembles the one supported by reflective component models such as FRACTAL [3]. In addition to the FRACTAL functionality, our framework automates the generation of most component interfaces and the management of the targeted system.

3 General-Purpose Autonomic Architecture

Figure 1 depicts our general-purpose autonomic architecture, a preliminary version of which was introduced in [5, 6]. The core component of the architecture is a universal policy engine that organises a heterogeneous collection of legacy ICT resources and autonomic-enabled resources into a self-managing system. To reduce

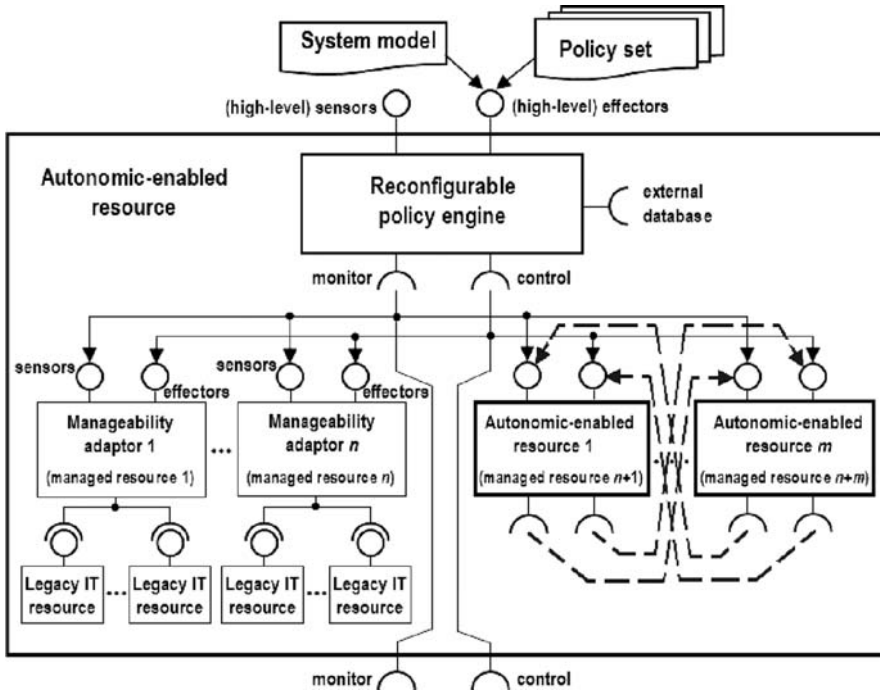


Fig. 1 UML component diagram of the autonomic architecture. The architecture supports the development of two types of autonomic systems-of-systems: a hierarchical topology that allows an instance of the policy engine to manage other instances of the architecture (i.e., the managed resources $n+1$ to $n+m$ in the diagram); and a federation of collaborating instances of the architecture that use each others' high-level sensors and effectors, as shown by the dashed lines in the diagram

the effort required to develop autonomic solutions, the policy engine can handle resources whose types are unknown during its implementation and deployment. This unique capability is achieved through runtime configuration: a *model* of the system to be managed is supplied to the policy engine for this purpose. As a result, the engine can implement the high-level goals described by a set of user-specified policies that make reference to the resources defined in the system model.

As recommended by IBM's architectural blueprint for autonomic computing [16], standardised adaptors are used to expose the *manageability* of all types of legacy ICT resources in a uniform way, through sensor and effector interfaces. The autonomic-enabled resources in the self-managing system are either typical ICT resources designed to expose sensor and effector interfaces allowing their direct inter-operation with the policy engine, or other instances of the architecture. The latter option is possible because the policy engine exposes the entire system as an atomic ICT resource through *high-level sensors* and *high-level effectors*. A detailed description of the architecture and an overview of existing standards and technologies that can be used to implement it in practice are available in [5, 6].

4 Reconfigurable Policy Engine

The internal architecture of our policy engine (Fig. 2) is influenced by the types of policies it implements and by its ability to handle resources whose characteristics are supplied to the engine at runtime. A “coordinator” module is employing the following components to implement the closed control loop of an autonomic system:

- The *runtime code generator* produces the necessary interfaces when the policy engine is configured to manage new types of resources or supplied with new resource-definition policies. When a new system model is used to configure the policy engine, *manageability adaptor proxies* are generated that allow the engine to interoperate with the manageability adaptors for the resource types specified in the system model. Likewise, when resource-definition policies are set up that specify new ways in which the policy engine should expose the ICT resources it manages, *high-level manageability adaptors* are generated.
- The *manageability adaptor proxies* are thin interfaces allowing the policy engine to communicate with the autonomic-enabled resources and the manageability adaptors for the legacy resources in the system.
- The *high-level manageability adaptors* expose the system state and configuration in a format that allows its integration within other instances of the architecture. The way in which these interfaces are dynamically specified by means of resource-definition policies is described later in the chapter.

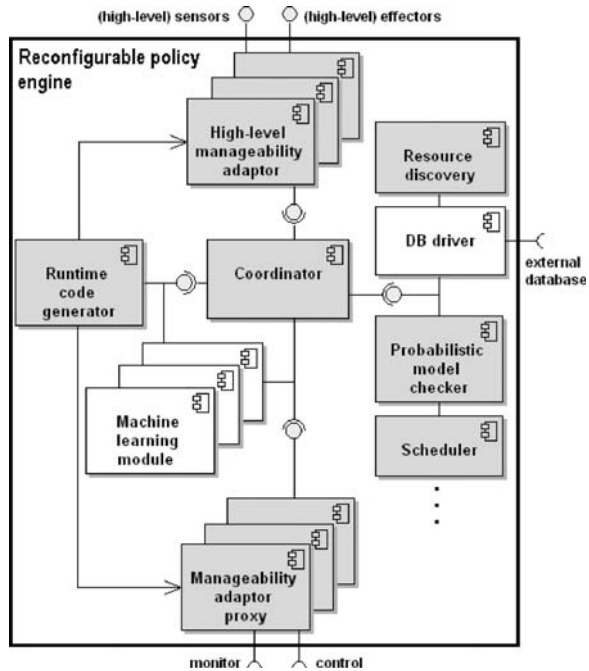


Fig. 2 Architecture of the reconfigurable policy engine. The shaded components are implemented by the prototype described in Sect. 5. A standards-based database driver will be added in a future version of the prototype. The machine learning modules represent the focus of ongoing research efforts by the autonomic computing community, and will be included in a reference implementation of the engine when the results of this research start to crystallise

- The *scheduler* is used to support the scheduling operators appearing in policy actions for the goal and utility-function policies handled by the policy engine.
- The *resource discovery* component is used to locate the resources to be managed by the policy engine.
- The *database driver* is used to maintain policy engine data such as historical resource property values in an external persistent storage.
- The *machine learning modules* use machine learning techniques [2] to derive and/or refine a behavioural model of the managed resources based on sensor data and inside policy engine information. This enables the engine to support goal and utility-function policies for systems for which in-depth knowledge about the behavioural characteristics of the managed resources cannot be supplied by system administrator. The usefulness of a *Modeler* component for the implementation of utility-function policies is mentioned in [44], although the authors are not specific about the learning algorithms that such a component might use.
- The *probabilistic model checker* enables the policy engine to take full advantage of the behavioural model supplied by the system administrator or built by its machine learning modules. This is done by using probabilistic model checking to establish quantitative properties of the system [24] and thus to implement the user-specified policies. As will be illustrated by a couple of the case studies in Sect. 7, the integration of these *quantitative verification* techniques into the policy engine enables system administrators to specify powerful goal and utility-function policies that would have been extremely complicated or even impossible to express otherwise. Another use envisaged for the model checker is to help verify the policies implemented by the engine as suggested in [22].

5 Prototype Implementation

In this section we overview a prototype implementation of our autonomic architecture that was originally introduced in [7], and we describe for the first time two of its new features: the integration of a probabilistic model checker with the policy engine and the implementation of resource-definition policies.

Two major choices influence the realisation of an instance of the architecture: the technology used to represent the system model and the technology chosen for the implementation of the policy engine components. We chose to represent system models as plain XML documents that are instances of a pre-defined meta-model encoded as an XML schema. This choice was motivated by the availability of numerous off-the-shelf tools for the manipulation of XML documents and XML schemas that are largely lacking for the other technologies we considered (e.g., [1, 29, 32]). In particular, by using existing XSLT engines and XML-based code generators, we shortened the prototype development time and avoided the need to implement bespoke components for this functionality.

As shown in Fig. 3, an ICT system is a named set of resources (*resource* in the UML diagram), each comprising a unique identifier *ID* and a set of resource

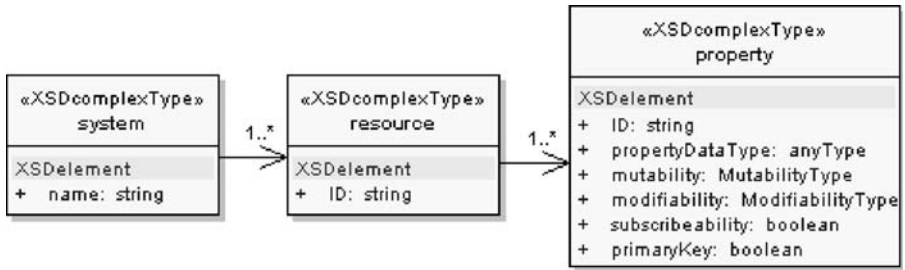


Fig. 3 Meta-model of an ICT system

properties with their characteristics. A resource property is associated a unique *ID*, and has a data type (i.e., *propertyDataType*). Several other property characteristics are defined in the meta-model:

- *mutability* – the WS-RMD MutabilityType [33] specifies if the property is “constant”, “mutable” or “appendable”;
- *modifiability* – tells if the property is “read-only”, “read–write”, “write-only” or “derived” from other properties and the behavioural model of the system;
- *subscribeability* – specifies whether a client such as the policy engine can subscribe to receive notifications when the value of this property changes;
- *primaryKey* – indicates whether the property is part of the property set used to identify a resource instance among all resource instances of the same type.

Our prototype policy engine and the manageability adaptors enabling its interoperation with legacy resources were implemented as web services in order to leverage the platform independence, loose coupling and security features of this technology [46]. The runtime configuration of the engine required the extensive use of techniques available only in an object-oriented environment, e.g., runtime generation of data types and manageability adaptor proxies, reflection and generics. Based on these requirements, J2EE and .NET were selected as candidate development platforms for the prototype engine, with .NET being eventually preferred due to its better handling of dynamic proxy generation and slightly easier-to-use implementation of reflection. The components included in the prototype are shown in Fig. 2.

The free, open-source probabilistic model checker PRISM [14] developed by the Quantitative Analysis and Verification Group at the University of Oxford was chosen for integration with the original version of the policy engine described in [7]. This choice was based on an extensive performance analysis of a range of model checkers [18] that ranked PRISM as the best option for analysing large behavioural models such as the ones encountered in autonomic computing systems. Furthermore, PRISM comes with a command-line interface that made possible its direct integration into the existing version of the policy engine, and the runtime execution of *quantitative analysis experiments* [23, 24] that self-managing systems can use to realise powerful goal and utility-function policies as illustrated in Sects. 7.3 and 7.4.

Another novel feature of the policy engine that we describe for the first time is its ability to handle resource-definition policies, i.e., policies of the form

$$\text{RESDEF}(\text{newResourceId}, \text{propertyDef}_1, \dots, \text{propertyDef}_m), \quad (1)$$

where *newResourceId* is a string corresponding to the *ID* element of a *resource* definition from the meta-model in Fig. 3 and

$$\text{propertyDef}_i = (\text{propertyId}_i, \text{expr}_i, \text{subscribeability}_i, \text{primaryKey}_i), 1 \leq i \leq m \quad (2)$$

define the properties of the new resource type. The expr_i component in (2) tells the policy engine how to calculate the value of the *i*th resource property as a function of the resources in the policy *scope*, or is one of `INTEGER`, `DOUBLE` or `STRING` to indicate that property *i* is a “read–write” property with one of these primitive types. The other components of propertyDef_i correspond to the property characteristics from the system meta-model in Fig. 3 that cannot be inferred from expr_i . To implement a resource-definition policy, the policy engine generates dynamically the data type for the new resource and its manageability adaptor (i.e., a new web service whose URL is built by replacing the suffix `PolicyEngine.asmx` from the policy engine URL with `newResourceIdManageabilityAdaptor.asmx`). This manageability adaptor exposes objects of the new data type that are created and whose fields are set in accordance with the property definitions (2). The case study presented in Sect. 7.5 illustrates the use of resource-definition policies.

6 A Generic Method for the Development of Autonomic Systems

Our method for the development of autonomic systems comprises four steps:

1. development of a model of the system to which autonomic capabilities are added;
2. generation of manageability adaptors for the legacy resources in the system;
3. reconfiguration of the policy engine by means of the system model from step 1;
4. development of autonomic computing policies that handle the required use cases.

To illustrate these steps, we will apply them to a system comprising a set of services of different priorities, subjected to different workloads, and sharing the CPU capacity of the same server. The aim of the case study is to develop an autonomic solution for managing the allocation of CPU to services such that high-priority services are treated preferentially, subject to each service getting a minimum amount of CPU.

Several policy types are typically used in autonomic systems [44, 45]: *action policies* provide a low-level specification of how the system configuration should be changed to match its state; *goal policies* specify precise constraints that should be met by varying the system configuration and *utility-function policies* supply a “measure of success” that the self-managing system should optimise by appropriately varying its configuration. In our running example we will use a utility-function policy, which is the most flexible of these policy types.

To implement utility-function policies, the policy engine needs an understanding of the *behaviour* of the system and its resources. Given a resource, we define its state \mathbf{s} as the vector whose elements are the read-only properties of the resource, and its configuration \mathbf{c} as the vector comprising its modifiable (i.e., read–write) properties. Let S and C be the value domains for \mathbf{s} and \mathbf{c} , respectively.² A behavioural model of the resource is a function

$$\textit{behaviouralModel} : S \times C \rightarrow S, \quad (3)$$

such that for any current resource state $\mathbf{s} \in S$ and for any resource configuration $\mathbf{c} \in C$, $\textit{behaviouralModel}(\mathbf{s}, \mathbf{c})$ represents the future state of the resource if its configuration is set to \mathbf{c} .

Our policy engine works both with an approximation of the behavioural model that consists of a set of discrete values of the *behaviouralModel* in (3) and with a continuous-time Markov chain (CTMC) [23] representation of (3). For our running example, we will use the former type of behavioural model; the use of CTMC behavioural models is described in Sect. 7. As the current version of the policy engine does not include the machine learning modules described in Sect. 4, it acquires these behavioural models from the manageability adaptors for the managed resources. With the future addition of machine learning modules (Fig. 2), the policy engine will gain the ability to use learning techniques to refine and, eventually, to derive these behavioural models automatically based on its observation of the managed resources.

Step 1: Model Development Let *System* be the set of all instances of the meta-model in Fig. 3; the purpose of this step is to find a system model

$$M \in \textit{System} \quad (4)$$

that can be used to implement the desired autonomic solution. To achieve this goal, we identify the system resources involved in the autonomic solution and their relevant properties. Given the ability to reconfigure the policy engine at any time, it makes sense to keep this model as simple as possible: additional resources and/or resource properties can be specified in new versions of the model, and conveyed to the policy engine as and when necessary. For instance, the single resource type for our example system is *service*, and its properties are as follows: *name*, a unique identifier used to distinguish between different services; *priority*, an integer value; *cpuAllocation*, the percentage of the server CPU allocated to the service; *responseTime*, the service response time, averaged over the past one-second time interval; *interArrivalTime*, the request inter-arrival time, averaged over the past one-second time interval and *behaviouralModel*, an approximation of the service behaviour that provides information on how the service response time varies with its CPU allocation and the request inter-arrival time.

² Note that S and C are fully specified in the system model.

Each resource property is then analysed in order to identify its value domain, mutability, modifiability and all of the other characteristics specified by the meta-model in Fig. 3. This information is encoded as an instance of the system meta-model, ready to be used in the subsequent steps of the method. By analysing these resource properties for our running example and representing the analysis results as an instance of the system meta-model, we produced with the system model in Fig. 4.

Step 2: Manageability Adaptor Generation Given a system model M , this step generates manageability adaptors for each type of legacy resource. Off-the-shelf tools can be used to automate most of this generation. First, an XSLT transformation

$$schemaGen : System \rightarrow XmlSchema \tag{5}$$

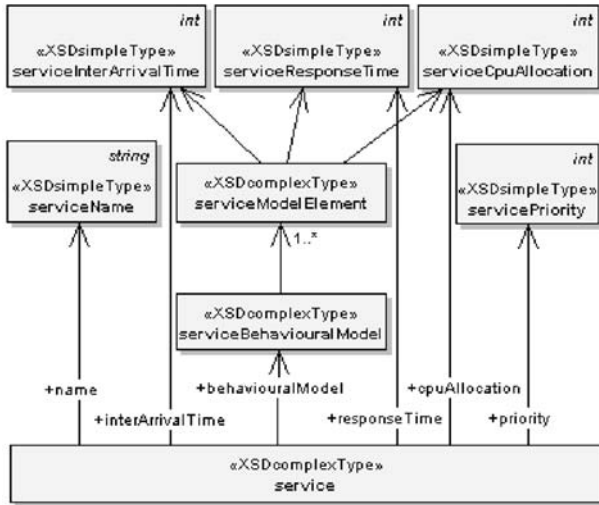
is applied to the system model in order to obtain an XML schema for the resource types in the system. The XML schema generated when this transformation is applied to our sample system model is depicted as UML in Fig. 5a. A standard data type

```

<system xmlns="...">
  <name>server</name>
  <!-- Services running within a server -->
  <resource>
    <ID>service</ID>
    <property>
      <ID>name</ID>
      <propertyDataType>
        <xs:simpleType name="serviceName">
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </propertyDataType>
      <mutability>constant</mutability>
      <modifiability>read-only</modifiability>
      <subscribeability>>false</subscribeability>
      <primaryKey>true</primaryKey>
    </property>
    <property>
      <ID>priority</ID>
      ...
    </property>
    <property>
      <ID>cpuAllocation</ID>
      <propertyDataType>
        <xs:simpleType name="serviceCpuAllocation">
          <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
          </xs:restriction>
        </xs:simpleType>
      </propertyDataType>
      <mutability>mutable</mutability>
      <modifiability>read-write</modifiability>
      <subscribeability>>false</subscribeability>
      <primaryKey>false</primaryKey>
    </property>
  </resource>
</system>
  <property>
    <ID>responseTime</ID>
    ...
  </property>
  <property>
    <ID>interArrivalTime</ID>
    ...
  </property>
  <property>
    <ID>behaviouralModel</ID>
    <propertyDataType>
      <xs:complexType
        name="serviceBehaviouralModel">
        <xs:sequence>
          <xs:element name="modelElement"
            type="serviceModelElement"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </propertyDataType>
    <xs:complexType name="serviceModelElement">
      <xs:sequence>
        <xs:element name="responseTime"
          type="serviceResponseTime"/>
        <xs:element name="interArrivalTime"
          type="serviceInterArrivalTime"/>
        <xs:element name="cpuAllocation"
          type="serviceCpuAllocation"/>
      </xs:sequence>
    </xs:complexType>
  </propertyDataType>
  <mutability>constant</mutability>
  <modifiability>read-only</modifiability>
  <subscribeability>>false</subscribeability>
  <primaryKey>false</primaryKey>
</property>

```

Fig. 4 System model for the running example



a



b

Fig. 5 Generated XML schema (a) and manageability adaptor (b) for the sample system

generator such as Microsoft’s XML Schema Definition tool [28] is then used to automatically generate the data type set associated with this schema:

$$data\ TypeGen : XmlSchema \rightarrow \mathbb{P} \text{DataType}. \tag{6}$$

Finally, a simple transformation was implemented to automate the generation of manageability adaptor stubs for the legacy resources in the system:

$$adaptorGen : XmlSchema \rightarrow \mathbb{P} ManageabilityAdaptor. \tag{7}$$

As shown in Fig. 5b, which depicts the data type (i.e., **service**) and the manageability adaptor (i.e., **ServiceManageabilityAdaptor**) for the system in our running example, all manageability adaptors are subclassing the generic abstract web service `ManagedResource<T>`. The bulk of the sensor and effector functionality associated with a manageability adaptor is implemented in this base abstract class, and only a small number of simple, resource-specific methods that are declared abstract in `ManagedResource<T>` need to be implemented manually in each manageability adaptor. Note that the policy engine is itself implemented as a subclass of `ManagedResource<T>`, so that an instance of the architecture can be readily included as a managed resource into a larger autonomic system as described in Sect. 3.

To complete this step, the manageability adaptor produced by the generator in (7) and depicted in Fig. 5b was manually extended, and then connected to a server discrete-event simulator running a high-priority “premium” service and a low-priority “standard” service. These services handled simulated requests with normally distributed CPU utilisation and exponentially distributed inter-arrival time.

Step 3: Engine Configuration This step consists in supplying the system model to the instance of the policy engine used in the autonomic solution. As stated before, the policy engine was realised as a web service, so we implemented a web interface for its simple configuration. Figure 6 shows a snapshot of this interface after the

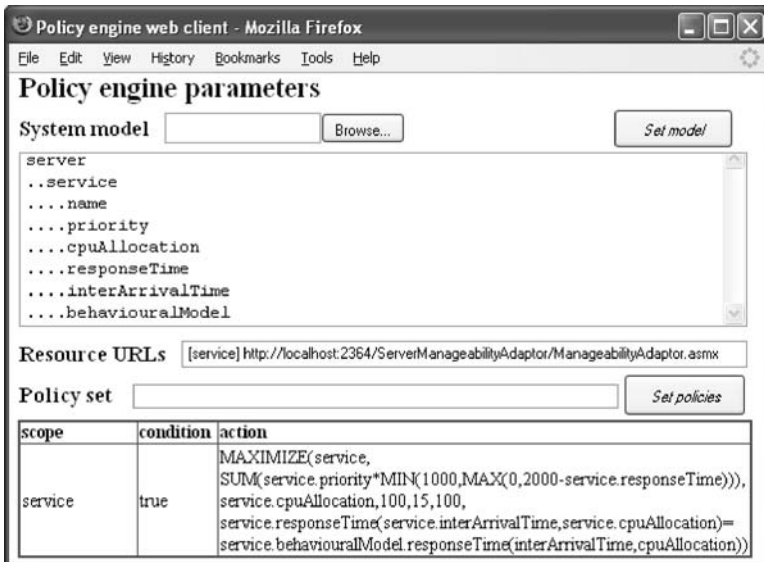


Fig. 6 Policy engine configuration

system model from our running example, and the utility-function policy that will be presented in step 4 were supplied to the engine.

Step 4: Policy Development In this step, autonomic computing policies are designed that support the use cases of the envisaged autonomic solution. The scope, priority, condition and action components of these policies make reference to the resources and resource properties defined in the system model used to configure the policy engine. Each of these policy components can be specified using a rich set of operators and functions [6] that allow the definition of action, goal, utility-function and, in the latest version of the engine, of resource-definition policies.

The policy set is applied to all resources whose locations are known to the policy engine,³ and which are in the scope of the policies. Policy development is generally a complex, error-prone and iterative process [4], and our framework improves the effectiveness of this process significantly by (a) enabling and encouraging the reuse of system models and policies and (b) simplifying the iterative development and testing of policies for new types of resources and of policies that explore the use of new properties of existing resources in novel ways.

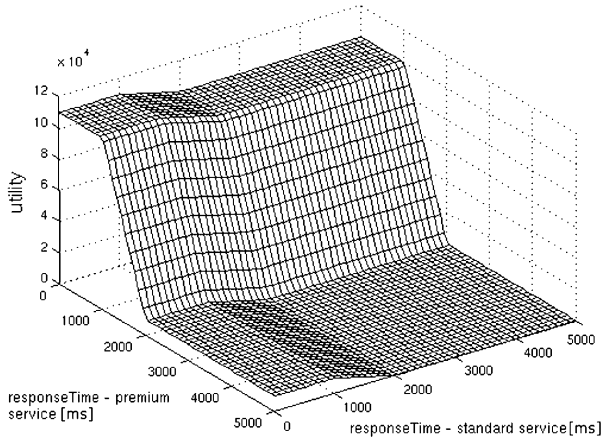
For our autonomic solution, we defined a utility function that models the business gain associated with running a set of `service` resources R with different levels of service:

$$utility(R) = \sum_{r \in R} r.priority * \min(1000, \max(0, 2000 - r.responseTime)).$$

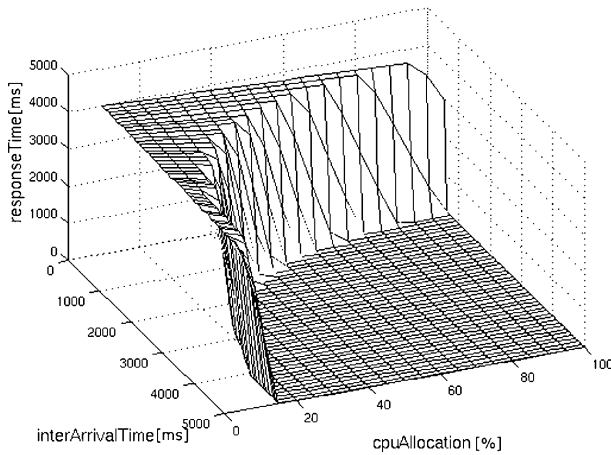
Figure 7a depicts the utility function for a server running a “premium” service with priority 100 and a “standard” service with priority 10. The policy action implemented by the autonomic system (Fig. 6 and Table 1) was defined by means of the `MAXIMISE(R , $utility$, $property$, $capacity$, min , max , $model$)` operator that uses the information about the system behaviour encoded in `model` to set the value of the specified resource `property` for all resources in R such as to: (a) maximise the value of the `utility` function and (b) ensure that the value of `property` stays between `min` and `max`, and that the sum of the `property` values across all resources in R does not exceed the available `capacity`.

This policy provides the definition of the utility function, and the link between the `responseTime`, `interArrivalTime` and `cpuAllocation` properties of a `service` resource and the components of its `behaviouralModel` property. Each time it evaluates the utility-function policy, the policy engine uses this information to select the elements from the behavioural model that are in the proximity of the current state of the system; the Euclidean metric is used for this calculation. The new configuration for the system is then chosen as the one associated with the selected element that maximises the value of the utility function. The experimental results of applying this policy to our example system are presented in Sect. 7.1.

³ The policy engine employs a resource discovery service (Fig. 2) to obtain the URLs of the resources to be managed.



a



b

Fig. 7 Utility function (a) and service behavioural model (b) for the running example

Table 1 The arguments of the MAXIMISE(*R*, *utility*, *property*, *capacity*, *min*, *max*, *model*) policy action for the running example of an autonomic system

<i>R</i>	<i>service</i>
<i>utility</i>	SUM(<i>service.priority</i> * MIN(1000, MAX(0, 2000 – <i>service.responseTime</i>)))
<i>property</i>	<i>service.cpuAllocation</i>
<i>capacity</i>	100
<i>min</i>	15
<i>max</i>	100
<i>model</i>	<i>service.responseTime</i> (<i>service.interArrivalTime</i> , <i>service.cpuAllocation</i>) = <i>service.behaviouralModel.responseTime</i> (<i>service.behaviouralModel.interArrivalTime</i> , <i>service.behaviouralModel.cpuAllocation</i>)

7 Case Studies

7.1 Utility-Driven Allocation of CPU Capacity

We start our presentation of case studies with the experimental results for the running example of an autonomic system from the previous section. Variants of this system were used to validate autonomic computing frameworks in the past (e.g., [44]), hence this well-understood use case provides a good basis for a first assessment of the framework. To evaluate our autonomic solution, the behavioural model for a service was obtained from 100 runs of the server simulator in which the average service response time was recorded for 920 equidistant points covering the entire (*interArrivalTime*, *cpuAllocation*) value domain (Fig. 7b). Figure 8 shows a typical experiment in which the utility-function policy in Table 1 was used to manage the allocation of CPU to our “premium” and “standard” services, when their request inter-arrival times were varied to simulate different workloads. The policy evaluation period was set to 3 s for this experiment, so that the system could self-adapt to the rapid variation in the workload of the two services. This allowed us to measure the CPU overhead of the policy engine, which was under 1% with the engine service running on a 1.8 GHz Windows XP machine. In a real scenario, such variations in the request inter-arrival time are likely to happen over longer intervals of time, and the system would successfully self-optimize with far less frequent policy evaluations.

7.2 Goal-Based Scheduling of CPU Capacity

In the absence of knowledge about the behaviour of the legacy ICT resources that need to be organised into a self-managing system, goal policies can often be used in conjunction with scheduling heuristics. In this section, we consider the same system as in Sect. 7.1, but assume that a behavioural model describing the variation of the service response time with its allocated CPU and request inter-arrival rate is not available. Figure 9 depicts a concise representation of the system model and a goal policy that can be used in this scenario. The action of this goal policy is specified by means of an expression that uses the `SCHEDULE(R, ordering, property, capacity, min, max, optimal)` operator that (a) sorts the resources in *R* in non-increasing order of the comparable expressions in *ordering*; (b) in the sorted order, sets the specified resource *property* to a value never smaller than *min* or larger than *max*, and as close to *optimal* as possible; and (c) ensures that the overall sum of all *property* values does not exceed the available *capacity*. Accordingly, the policy action in Fig. 9 will set the *cpuAllocation* property of all services to a value between 15 and 100%, subject to the overall CPU allocation staying within the 100% available capacity. Optimally, the *cpuAllocation* should be left unchanged if the $55 \leq \text{cpuUtilisation} \leq 85$,

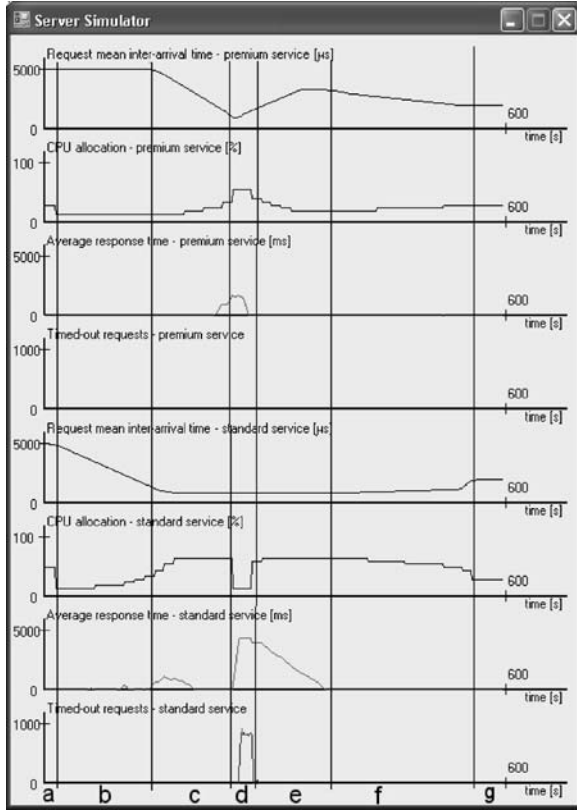


Fig. 8 Experimental results for Sect. 7.1. The CPU allocations for the services are initially decreased to match their light workload (5 ms request inter-arrival time during time interval a). As the service workloads increase, so do the CPU allocations, until the CPU required to satisfy the demand from the premium service leaves insufficient CPU capacity for the standard service to make any contribution to the utility function (time interval d), hence it is allocated the minimum amount of CPU specified in the policy (i.e., 15%). As soon as less CPU capacity is required to satisfy the needs of the premium service (time interval e), the standard service is swiftly allocated sufficient CPU to bring it back into a region of operation in which it contributes to the utility function. Subsequently, the CPU allocations are varied to accommodate more gradual changes in the workloads (time intervals f–g)

decrease by 5% if $cpuUtilisation < 55$ and increase by 5% if $cpuUtilisation > 85$.⁴ The experimental results for the resulting autonomic solution (available in [7]) resemble those corresponding to the use of a utility-function policy in Sect. 7.1, but are less effective in two important circumstances:

⁴ The $HYSTERESIS(val, lower, upper)$ operator used to achieve this behaviour (Fig. 9) returns -1 , 0 or 1 if $val < lower$, $lower \leq val \leq upper$ or $upper < val$, respectively.

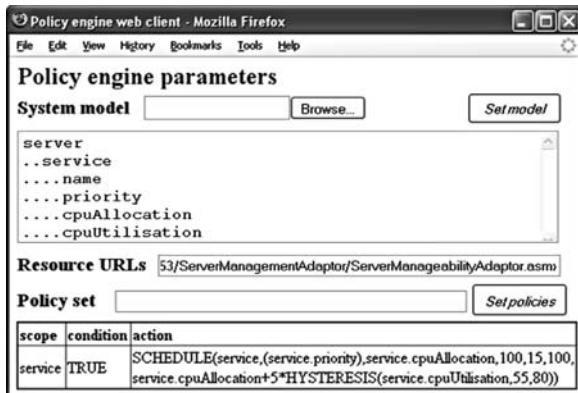


Fig. 9 Policy engine parameters for the case study in Sect. 7.2. The policy engine is configured to monitor the service `cpuUtilisation` (i.e., the amount of CPU utilised by the service, expressed as a percentage of its CPU allocation) and to realise a goal policy requiring that the `cpuUtilisation` is maintained between 55 and 80% of the allocated CPU

- several successive policy evaluations are required to handle significant changes in the service workloads because the CPU capacity allocated to services can be modified by only $\pm 5\%$ at a time;
- when insufficient CPU is available to ensure that a low-priority service runs in an operation area that is useful for the business and the utility-function policy in Sect. 7.1 would restrict the CPU allocated to the service to a minimum, the goal policy gives it all available CPU, thus wasting CPU capacity unnecessarily.

7.3 Dynamic Power Management of Disk Drives

When formal methods are used in the development and/or verification of legacy ICT resources, the behavioural models employed by these methods can often be exploited by our framework to augment the legacy ICT resources with autonomic capabilities. Starting from the CTMC model of a Fujitsu disk drive in [38] and its encoding as a PRISM CTMC model [37], we built (Fig. 10) a system model of the disk drive that can be used for the configuration of our policy engine. We then used this system model to add self-optimisation capabilities to the disk drive so that it dynamically adapted its probability of transitioning from the *idle* state to the low-power *sleep* state to changes in (a) the request inter-arrival time and (b) the user-specified utility function:

$$utility = w_1 \min \left(1, \max \left(0, \frac{11 - queueLength}{2} \right) \right) + w_2 \max(0, 1.2 - power), \quad (8)$$

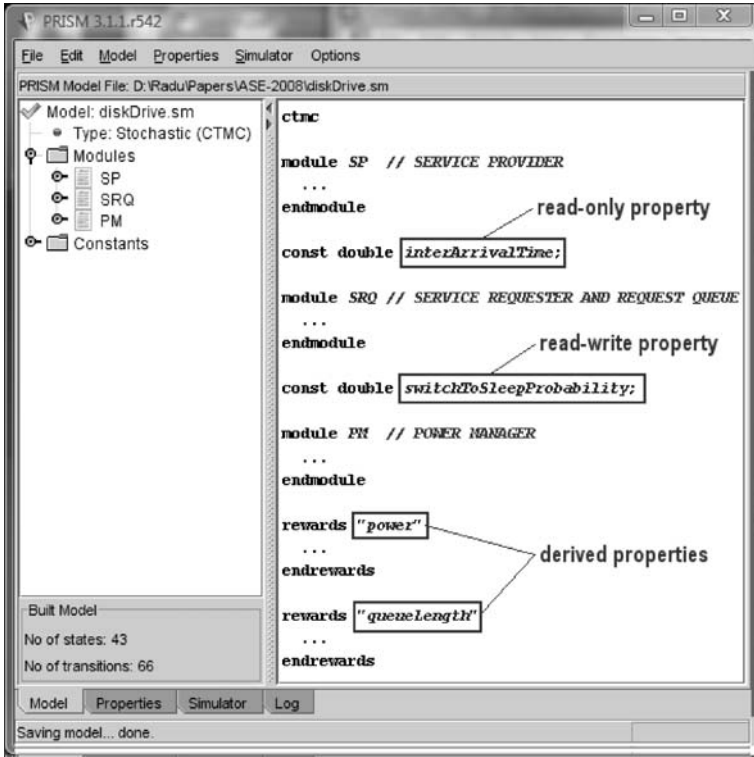


Fig. 10 PRISM CTMC model of a three-state Fujitsu disk drive taken from [37], and used to devise the system model for the configuration of the policy engine. The uninitialised PRISM constants correspond to “read-only” and “read-write” properties of a disk drive resource (i.e., `interArrivalTime` and `switchToSleepProbability`, respectively). PRISM reward structures (i.e., `power` and `queueLength`) correspond to “derived” disk drive properties

where the weights w_1 and w_2 are chosen depending on the circumstances in which the disk drive is used (Fig. 11). Given this policy, the policy engine ran PRISM experiments [24] to establish the optimal `switchToSleepProbability` for the disk drive at regular, 10-s time intervals. For our simple CTMC model, each of these experiments took subsecond time, yielding the results in Fig. 12.

7.4 Adaptive Control of Cluster Availability

The case study presented in this section involves the adaptive control of cluster availability within a data centre. The aim of the autonomic solution is to control the number of servers allocated to the $N \geq 1$ clusters of a data centre in order to maximise the utility function

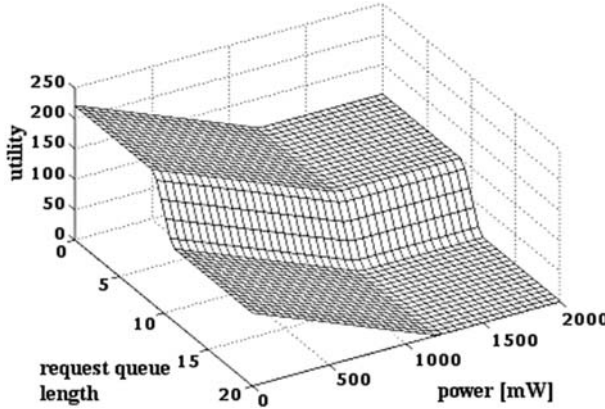


Fig. 11 The utility function (8) (depicted here for $w_1 = w_2 = 100$) was used to achieve a user-customisable trade-off between the disk drive responsiveness (which is probably proportional to its average queueLength [38]) and its power consumption (i.e., power)

$$utility = \sum_{i=1}^N priority_i \cdot GOAL(availability_i \geq target_availability_i) - \epsilon \sum_{i=1}^N servers_i \quad (9)$$

subject to

$$\sum_{i=1}^N servers_i \leq Total_servers \text{ and } required_i \leq servers_i, \quad (10)$$

where $priority_i > 0$, $availability_i \in [0, 1]$, $target_availability_i \in [0, 1]$, $required_i \geq 1$ and $servers_i \geq 1$ represent the priority, (actual) availability, target availability, number of required servers and number of (allocated) servers for cluster i , $1 \leq i \leq N$, respectively. The GOAL operator yields 1 when its argument is true and 0 otherwise, $Total_servers \geq 1$ is the total number of servers in the data centre, and $0 < \epsilon \ll 1$ is a constant.⁵ The availability of cluster i , $availability_i$, is the fraction of a 1-year time period during which at least $required_i$ servers are usable (i.e., they are operational and connected to an operational switch and backbone).

Like in the previous case study, we extracted the system model for the configuration of our policy engine from an existing behavioural model of the targeted ICT resource, namely from the CTMC model of a dependable cluster of workstations introduced in [11]. This model takes into account the failure and repair rates of all components from our targeted cluster architecture (Fig. 13a). Consequently, the policy engine can use PRISM to calculate the cluster availabilities for the data-centre configurations satisfying (10), and to decide the number of servers that each cluster

⁵ The second term of the utility function (9) ensures that when multiple configurations maximise the first term, the configuration that uses the fewest servers is preferred.

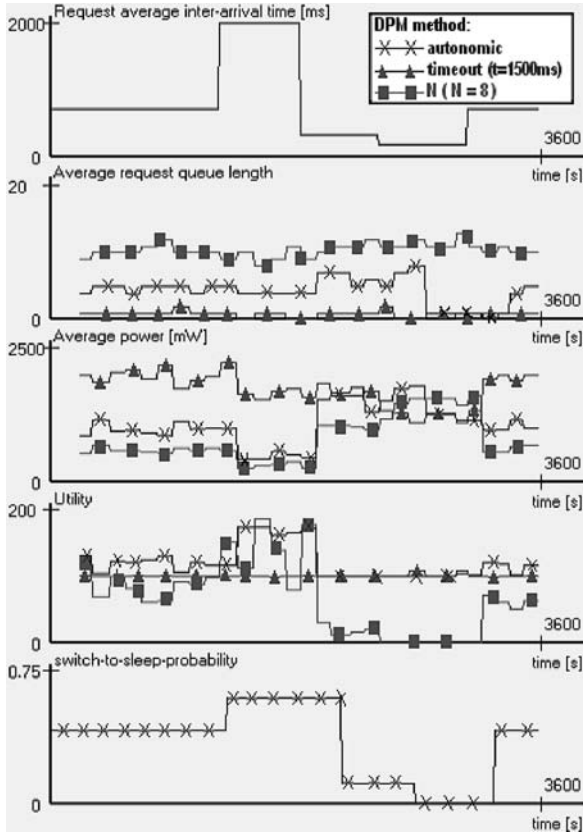


Fig. 12 Discrete-event simulation results contrasting our autonomic approach to disk drive dynamic power management (DPM) with two standard DPM methods [38]: the *timeout method* that moves the disk drive into the sleep state after a period of idleness t and “awakens” it immediately after a request has arrived; and the *N method* that moves the disk drive into the sleep state as soon as it becomes idle, and “awakens” it after N requests accumulate in its queue. The autonomic DPM approach achieved a better utility than the two standard DPM methods for most of the time, and similar utility to the better of the two for the rest of the time. This is due to the good trade-off that the autonomic approach realised between power consumption and request queue length across a wide workload range, while the other approaches are effective for specific workloads

should get so that the value of the utility function (9) is maximised. Given the complexity of the CTMC behavioural model, we implemented a cluster manageability adaptor that uses notifications to inform the policy engine about changes in the number of required servers for the clusters. Hence, the policy engine recalculates the server allocations only when there is a change in the state of the autonomic system. In our simulations, this calculation took up to 30 s. This response time is acceptable for the considered use case because, based on our previous experience with policy-based data-centre management [4], half a minute represents a small

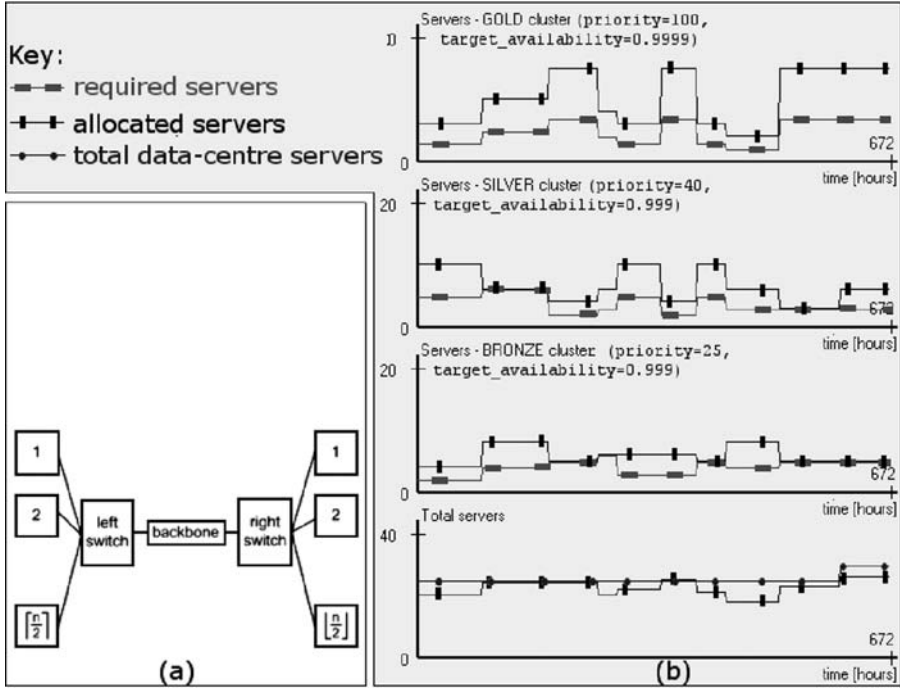


Fig. 13 Architecture of an n -server dependable cluster, taken from [11] (a), and simulation results for a three-cluster data centre over a 4-week time period (b)

delay compared to the time required to provision a server when it is allocated to a new cluster.⁶ The experimental results are shown in Fig. 13b.

7.5 Dynamic Web Content Generation

The last case study is extending the autonomic solution from the previous section by incorporating the autonomic system for controlling cluster availability into an autonomic system of systems (Fig. 14). The resource-definition policy action below was supplied to policy engine instances within the autonomic data-centre systems:

```
RESDEF(businessValue, (id, CONCAT(cluster.id), false, true),
(max, SUM(cluster.priority), true, false), (actual, SUM(cluster.priority*
GOAL(cluster.availability >= cluster.targetAvailability)), true, false)).
```

(11)

⁶ Section 8 suggests techniques for working around the time taken by runtime model checking when such delays are not acceptable.

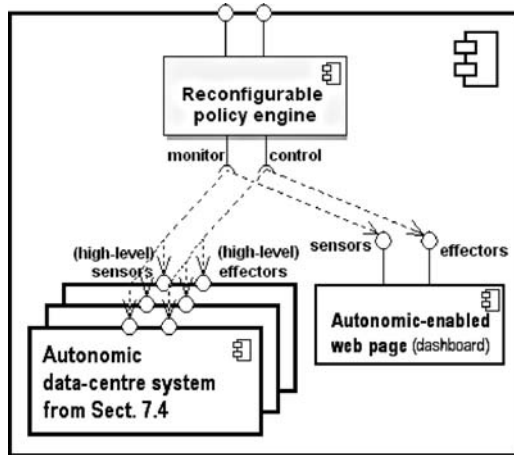


Fig. 14 Autonomic system of systems comprising several instances of the data-centre system from Sect. 7.4, and an autonomic-enabled web page implementing a business dashboard. The data-centre systems were each configured to expose their actual and ideal utility by means of a resource-definition policy, and the top-level policy engine implements an action policy that updates the properties of the autonomic-enabled web page with a summary of these utilities

As described in Sect. 5, this resulted in each of these policy engines dynamically creating a new ICT resource named `businessValue` and comprising three “read-only” properties: `id` – the concatenated identifiers of its clusters; `max` – its ideal utility, i.e., the maximum possible value of the first term in (9); and `actual` – the actual value of this term. A model of this synthesised ICT resource and of an autonomic-enabled web page was then used to configure the top-level policy engine in Fig. 14, and an action policy was used to ensure that this policy engine updates the web page periodically with a summary based on the `businessValue` of each autonomic data-centre system it knows about (Fig. 15).

8 Summary and Future Work

The success of mainstream computing is largely due to the availability of a system development methodology that enables and encourages standardisation, component reuse and user adoption. Building on recent advances in autonomic computing and on our previous work on policy-based autonomic systems, we proposed a general-purpose framework that brings similar benefits to the realm of autonomic computing. We introduced a set of criteria for assessing the generality of autonomic computing frameworks, and a new method for the development of self-managing systems starting from a model of their ICT resources. Also, we presented the integration of a probabilistic model checker into an autonomic computing policy engine, and we described how a new policy type termed a *resource-definition policy* can be used to build autonomic systems of systems.

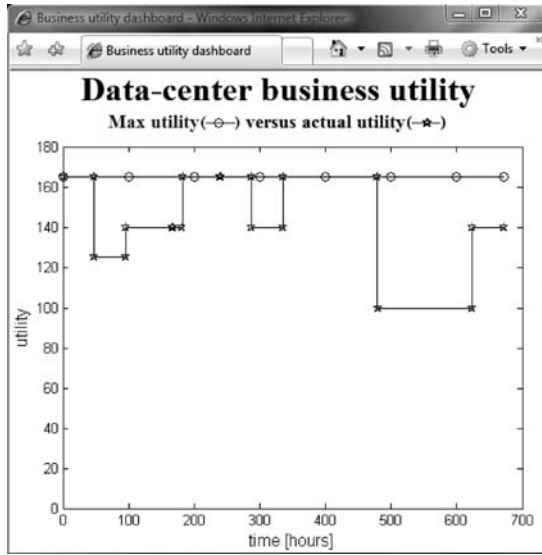


Fig. 15 An autonomic-enabled web page exposes effectors that the top-level policy engine uses to supply it with summary information about the maximum utility and actual utility of a set of autonomic data-centre systems (a single data-centre system was used in the experiment shown here). The web page presents the dynamically acquired information using a graphical representation that is generated at runtime using Matlab. Thus, the information about potential loss of business value is conveyed in a concise format that can be used directly by a data-centre manager

To validate our framework, we employed it to build autonomic solutions spawning a range of application domains and using a variety of autonomic computing policies. Table 2 uses these case studies to analyse the extent to which the proposed framework satisfies the generality criteria C1–C3 introduced in Sect. 1:

Table 2 Summary of the case studies presented in the paper

	C1 ICT resources				C2 self-* areas & policies				C3 application domain	
	Software	Hardware	Data	Legacy	Autonomic-enabled	Main self-* functional areas	Action	Goal	Utility-function	Resource-definition
Sect. 7.1	✓			✓		Self-monitoring Self-optimisation		✓		CPU capacity allocation
Sect. 7.2	✓			✓		Self-monitoring Self-optimisation	✓			CPU capacity allocation
Sect. 7.3		✓		✓		Self-monitoring Self-adaptation		✓		Dynamic power management
Sect. 7.4		✓		✓		Self-configuration Self-protection		✓		Cluster availability control
Sect. 7.5	✓	✓	✓	✓		Self-monitoring Self-generation	✓	✓	✓	Dynamic gen. of web content

- C1 In terms of supported ICT resources, our case studies demonstrate that the framework can handle the whole range of envisaged ICT resources.
- C2 The framework has been used to develop autonomic solutions in several areas of self-* functionality, and to support all types of autonomic computing policies. To further confirm its generality, new applications are being currently investigated that address additional areas of self-* functionality.
- C3 The autonomic systems developed for the presented case studies cover a range of application domains, including the development of a hierarchical system of systems. This is a good first step towards establishing that the framework satisfies this criterion. More work is required to assess the feasibility of using the framework in other use cases, and in particular in the development of federations of collaborating autonomic systems with no centralised management.

Based on past experience in using a domain-specific autonomic framework [4] to develop systems similar to those in Sects. 7.1 and 7.2, we estimate that the use of the generic framework to build these systems reduced the development effort by roughly an order of magnitude, and we expect the same to hold true for other applications.

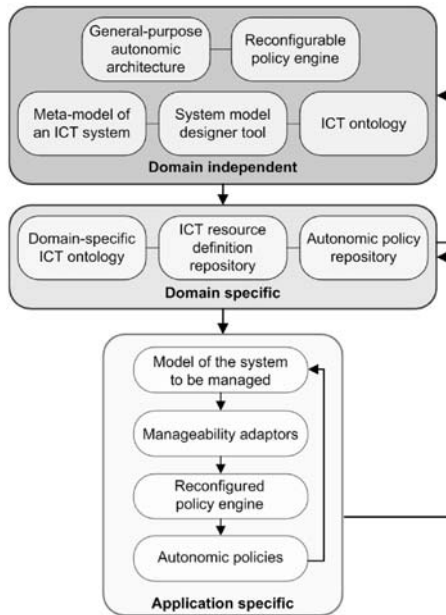


Fig. 16 Proposed autonomic system development methodology. The autonomic architecture, policy engine and system meta-model described in this paper are used at the domain-independent level, alongside a proposed ICT ontology and a proposed tool for designing the meta-model instances used to configure the policy engine. Repositories of ICT resource definitions and autonomic policies, and domain-specific ICT ontologies should be available at the level of an application domain, while our generic method for autonomic system development is employed for the cost-effective development of autonomic systems at the application-specific level

A key feature of our autonomic computing framework is its use of runtime probabilistic model checking. As shown in Sect. 7.4, model checking large systems can incur significant overheads, and the use of the subscription-notification mechanism supported by the framework (instead of periodical policy evaluation) is one way to accommodate this constraint. Other approaches to be investigated include the use of caching and pre-evaluation techniques to bypass the model checking step during policy evaluation, and the use of a hybrid approach in which a smaller model checking experiment is carried out to produce a close-to-optimal configuration for the autonomic system and a faster technique is then used to refine this configuration.

In addition to reusing components and techniques across a broad range of applications, our approach to autonomic system development allows and encourages the reuse of system models and autonomic computing policies. To take reusability further, these models and policies should draw their elements from domain-specific repositories of resource definitions and autonomic computing policies, respectively. Furthermore, to maximise the sharing of models, policies, manageability adaptors and autonomic-enabled resources, these repositories need to be built around controlled ICT ontologies, as required by the methodology for the cost-effective development of autonomic systems that we are proposing in Fig. 16. This methodology that we are working towards is in line with the excellent principles stated in [43] and successfully applied in the context of autonomic networking by Strassner et al. [42].

Acknowledgments The work presented in this chapter was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/F001096/1. The author is grateful to Marta Kwiatkowska, David Parker, Gethin Norman and Mark Kattenbelt for insightful discussions during the integration of the PRISM probabilistic model checker with the autonomic policy engine.

References

1. J. Arwe et al. Service Modeling Language, version 1.0, March 2007. <http://www.w3.org/Submission/2007/SUBM-sml-20070321>.
2. C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.
3. E. Bruneton et al. The FRACTAL component model and its support in Java. *Softw. Pract. Exper.*, 36:1257–1284, 2006.
4. R. Calinescu. Challenges and best practices in policy-based autonomic architectures. In *Proc. 3rd IEEE Intl. Symp. Dependable, Autonomic and Secure Computing*, pages 65–74, 2007.
5. R. Calinescu. Model-driven autonomic architecture. In *Proc. 4th IEEE Intl. Conf. Autonomic Computing*, June 2007.
6. R. Calinescu. Towards a generic autonomic architecture for legacy resource management. In K. Elleithy, editor, *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, pages 410–415, Springer, 2008.
7. R. Calinescu. Implementation of a generic autonomic framework. In D. Greenwood et al., editor, *Proc. 4th Intl. Conf. Autonomic and Autonomous Systems*, pages 124–129, March 2008.
8. M. Devarakonda et al. Policy-based autonomic storage allocation. In *Self-Managing Distributed Systems*, volume 2867 of *LNCS*, pages 143–154. Springer, 2004.
9. S. Dobson et al. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259, December 2006.

10. D. Gracanin et al. Towards a model-driven architecture for autonomic systems. In *Proc. 11th IEEE Intl. Conf. Engineering of Computer-Based Systems*, pages 500–505, 2004.
11. B. Haverkort et al. On the use of model checking techniques for dependability evaluation. In *Proc. 19th IEEE Symp. Reliable Distributed Systems*, pages 228–237, October 2000.
12. M. Hinchey et al. Modeling for NASA autonomous nano-technology swarm missions and model-driven autonomic computing. In *Proc. 21st Intl. Conf. Advanced Networking and Applications*, pages 250–257, 2007.
13. M.G. Hinchey and R. Sterritt. Self-managing software. *Computer*, 39(2):107–109, Feb. 2006.
14. A. Hinton et al. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th Intl. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
15. IBM Corporation. Autonomic computing: IBM’s perspective on the state of information technology, October 2001.
16. IBM Corporation. An architectural blueprint for autonomic computing, 2004. http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf.
17. IBM Corporation. Autonomic integrated development environment, April 2006. <http://www.alphaworks.ibm.com/tech/aide>.
18. D.N. Jansen et al. How fast and fat is your probabilistic model checker? An experimental comparison. In K. Yorav, editor, *Hardware and Software: Verification and Testing*, volume 4489 of *LNCS*, pages 69–85. Springer, 2008.
19. G. Kaiser et al. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *Proc. of the 5th Annual Intl. Active Middleware Workshop*, June 2003.
20. H. Kasinger and B. Bauer. Towards a model-driven software engineering methodology for organic computing systems. In *Proc. 4th Intl. Conf. Comput. Intel.*, pages 141–146, 2005.
21. J.O. Kephart and D.M. Chess. The vision of autonomic computing. *IEEE Computer Journal*, 36(1):41–50, January 2003.
22. S. Kikuchi et al. Policy verification and validation framework based on model checking approach. In *Proc. 4th IEEE Intl. Conf. Autonomic Computing*, June 2007.
23. M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. Foundations of Software Engineering*, pages 449–458. ACM Press, September 2007.
24. M. Kwiatkowska et al. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM’07)*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007.
25. C. Lefurgy et al. Server-level power control. In *Proc. 4th IEEE Intl. Conf. Autonomic Computing*, June 2007.
26. T. Lenard and D. Britton. *The Digital Economy Factbook*. The Progress and Freedom Foundation, 2006.
27. Wen-Syan Li et al. Load balancing for multi-tiered database systems through autonomic placement of materialized views. In *Proc. 22nd IEEE Intl. Conf. Data Engineering*, April 2006.
28. Microsoft Corporation. Xml schema definition tool (xsd.exe), 2007. [http://msdn2.microsoft.com/en-us/library/x6c1kb0s\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/x6c1kb0s(VS.80).aspx).
29. Microsoft Corporation. System Definition Model overview, April 2004. <http://download.microsoft.com/download/b/3/8/b38239c7-2766-4632-9b13-33cf08fad522/sdmwp.doc>.
30. B. Moore. Policy Core Information Model (PCIM) extensions, January 2003. IETF RFC 3460, <http://www.ietf.org/rfc/rfc3460.txt>.
31. R. Murch. *Autonomic Computing*. IBM Press, 2004.
32. B. Murray et al. Web Services Distributed Management: MUWS primer, February 2006. OASIS WSDM Committee Draft, <http://www.oasis-open.org/committees/download.php/17000/wsdm-1.0-muws-primer-cd-01.doc>.
33. OASIS. Web Services Resource Metadata 1.0, November 2006.

34. M. Parashar and S. Hariri. *Autonomic Computing: Concepts, Infrastructure & Applications*. CRC Press, 2006.
35. J. Parekh et al. Retrofitting autonomic capabilities onto legacy systems. *Cluster Computing*, 9(2):141–159, April 2006.
36. J. Pena et al. A model-driven architecture approach for modeling, specifying and deploying policies in autonomous and autonomic systems. In *Proc. 2nd IEEE Intl. Symp. Dependable, Autonomic and Secure Computing*, pages 19–30, 2006.
37. PRISM Case Studies: Dynamic Power Management. <http://www.prismmodelchecker.org/casestudies/power.php>.
38. Q. Qiu et al. Stochastic modeling of a power-managed system: construction and optimization. In *Proc. Intl. Symp. Low Power Electronics and Design*, pages 194–199. ACM Press, 1999.
39. M. Rohr et al. Model-driven development of self-managing software systems. In *Proc. 9th Intl. Conf. Model-Driven Engineering Languages and Systems*. Springer, 2006.
40. R. Sterritt et al. Sustainable and autonomic space exploration missions. In *Proc. 2nd IEEE Intl. Conf. Space Mission Challenges for Information Technology*, pages 59–66, 2006.
41. R. Sterritt and M.G. Hinchey. Biologically-inspired concepts for self-management of complexity. In *Proc. 11th IEEE Intl. Conf. Engineering of Complex Computer Systems*, pages 163–168, 2006.
42. J. Strassner et al. Providing seamless mobility using the FOCALE autonomic architecture. In *Proc. 7th Intl. Conf. Next Generation Teletraffic and Wired/Wireless Advanced Networking*, volume 4712 of *LNCS*, pages 330–341, 2007.
43. J. Strassner et al. Ontologies in the engineering of management and autonomic systems: A reality check. *Journal of Network and Systems Management*, 15(1):5–11, 2007.
44. W.E. Walsh et al. Utility functions in autonomic systems. In *Proc. 1st Intl. Conf. Autonomic Computing*, pages 70–77, 2004.
45. S.R. White et al. An architectural approach to autonomic computing. In *Proc. 1st IEEE Intl. Conf. Autonomic Computing*, IEEE Computer Society, pages 2–9, 2004.
46. O. Zimmermann et al. *Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer, 2005.

Software Architecture-Based Self-Adaptation

David Garlan, Bradley Schmerl, and Shang-Wen Cheng

Abstract Increasingly, systems must have the ability to self-adapt to meet changes in their execution environment. Unfortunately, existing solutions require human oversight, or are limited in the kinds of systems and the set of quality-of-service concerns they address. Our approach, embodied in a system called Rainbow, uses software architecture models and architectural styles to overcome existing limitations. It provides an engineering approach and a framework of mechanisms to monitor a target system and its environment, reflect observations into a system's architecture model, detect opportunities for improvement, select a course of action, and effect changes in a closed loop. The framework provides general and reusable infrastructures with well-defined customization points, allowing engineers to systematically customize Rainbow to particular systems and concerns.

1 Introduction

Imagine a world where a software engineer could take an existing software system and specify an objective, conditions for change, and strategies for adaptation to make that system self-adaptive where it was not before. Furthermore, imagine that this could be done in a few weeks of effort and be sensitive to maintaining business goals and other properties of interest. For example, an engineer might take an existing client-server system and make it self-adaptive with respect to a specific performance concern such as high latency. He might specify an objective to maintain request-response latency below some threshold, a condition to change the system if the latency rises above the threshold, and a few strategies to adapt the system to fix the high-latency situation. Another engineer might make a coalition-of-services system self-adaptive to network performance fluctuations, while limiting cost of operating the infrastructure. Still another engineer might make a cluster of servers self-adaptive to certain security attacks.

D. Garlan (✉)
Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
e-mail: garlan@cs.cmu.edu

Today, when increasingly systems have the requirement to self-adapt with minimal human oversight, it is becoming necessary to meet this vision. Systems must cope with variable resources, system errors, and changing user priorities, while maintaining, as best they can, the goals and properties envisioned by the engineers and expected from the users. Software engineers lack the tools and techniques to engineer a system with self-adaptation.

Engineers and researchers alike have responded to and met this self-adaptation need in somewhat limited forms through programming language features such as exceptions and in algorithms such as fault-tolerant protocols. But these mechanisms are often specific to the application, tightly bound to the code, and usually provide only localized treatment of system errors. As a result, self-adaptation for today's systems are costly to build, often taking many man-months to retrofit systems.

In contrast, the vision outlined above requires an approach that makes it possible for engineers to easily define adaptation policies that are global in nature, and that take into consideration business goals and quality attributes. In particular, we require that engineers be able to augment existing systems to be self-adaptive without rewriting them from scratch, that self-adaptation policies and strategies can be reused across similar systems, that multiple sources of adaptation expertise can be synergistically combined, and that all of this can be done in ways that support maintainability, evolution, and analysis.

In this chapter, we describe an approach to achieving these goals using architecture-based self-adaptation techniques. In particular, our approach abstracts observed behavior of an executing system into properties of an architectural model, where they can be reasoned about using a variety of existing architectural analysis techniques. The results of these analyses can then be used to reason about changes that should be made to a system to improve or correct the system's achievement of the quality attributes.

Our approach is embodied in a system called Rainbow, which focuses on two challenges to achieve cost-effective self-adaptation: (1) an approach and mechanism that reduces engineering effort and (2) representation of adaptation knowledge. Rainbow provides an engineering approach and a framework of mechanisms to monitor a system and its executing environment, reflect observations into an architectural model of the system, determine any problem states, select a course of action, and effect changes. By leveraging the notion of architectural style to exploit commonality of systems, the framework provides a general and reusable infrastructure with well-defined customization points to cater to a wide range of systems. The framework also provides a set of abstractions that allow engineers to focus on adaptation concerns, facilitating an adaptation engineering workflow for the systematic customization of Rainbow. To emulate the mundane and routine adaptation tasks performed by system administrators, Rainbow provides a language, called Stitch, to represent the adaptation techniques using first-class adaptation concepts. It offers modularity with respect to quality dimension and domain expertise, strategies with condition and effect, a mechanism to tailor to particular styles, and the use of utility theory to compute the best adaptation path under uncertainty.

In this chapter, we introduce the ideas behind architecture-based self-adapting systems; briefly survey the research landscape; discuss the research and engineering challenges, particularly with respect to autonomic behavior for distributed, networked systems; and describe the Rainbow approach and how it addresses these challenges. We also give examples of its use in the context of autonomic networks, focusing on adaptations to improve qualities such as fidelity, performance, security, and cost of operation.

2 Overview of Autonomic and Self-Adaptive Systems

Overcoming the challenges of self-adaptation and allowing managed systems to self-adapt with minimal human oversight requires closing the “loop of control.” Software systems have traditionally been designed as *open-loop* systems: once a system is designed for a certain function and deployed, its extra-functional quality attributes typically remain relatively unchanged. In most cases, if something goes wrong, humans must intervene, often by restarting the failed subsystem or taking the entire system offline for repair. This results in high costs in system downtime, personnel costs, and decreased revenue through system unavailability.

To address this problem, a number of researchers have proposed an alternative approach that uses external software mechanisms to maintain a form of closed-loop control over the target system (e.g., [26, 30, 39]). Such mechanisms allow a system to self-adapt dynamically, with reduced human oversight. Minimally, closed-loop control consists of mechanisms that monitor the system, reflect on observations for problems, and control the system to maintain it within acceptable bounds of behavior. This kind of system is known as a *feedback control system* in control theory [42].

Feedback control systems have typically been applied to control physical systems. For simple systems, the control model may be built-in to the design. For example, a home thermostat that measures room temperature and checks it against the set point, controlling a home heating and cooling system, will typically have a simple built-in thermodynamic model. In more complex systems an explicit process model is necessary for effective control [42]. For example, an air conditioning system for a large building that monitors and controls multiple locations would require an explicit model of the building partitions and temperatures to efficiently control which cooling units to turn on and when.

For software systems, the external controller requires an explicit model of the target system in order to reflect on observations and to configure and repair the system [39]. Monitoring mechanisms extract and aggregate target system information to update the model. An evaluation mechanism detects problems in the target system as reflected in the model. The appearance of a problem triggers an adaptation mechanism to use the model to determine a course of action. The mechanism then propagates the necessary changes to the target system to fix the problem.

In principle, external mechanisms have a number of benefits over internal mechanisms. External control separates the concerns of system functionality from those of

adaptation (or “exceptional”) behaviors. With the adaptation mechanism as a separate entity, engineers can more easily modify and extend it, and reason about its adaptation logic. Furthermore, the separation of mechanisms allows the application of this technique even to legacy systems with inaccessible source code, as long as the target system provides, or can be instrumented to provide, hooks to extract system information and to make changes. Finally, providing external control with generic but customizable mechanisms (e.g., model management, problem detection, strategy selection) facilitates reuse across systems, reducing the cost of developing new self-adaptive systems.

2.1 The IBM Autonomic Framework

The IBM Autonomic Computing Initiative codified an external, feedback control approach in its Autonomic Monitor-Analyze-Plan-Execute (MAPE) Model [28]. Figure 1 illustrates the MAPE loop, which distinguishes between the *autonomic manager* (embodied in the large rounded rectangle) and the *managed element*, which is either an entire system or a component within a larger system. The MAPE loop highlights four essential aspects of self-adaptation:

1. **Monitor:** The monitoring phase is concerned with extracting information—properties or states—out of the managed element. Mechanisms range from source-code instrumentation to non-intrusive communication interception.
2. **Analyze:** is concerned with determining if something has gone awry in the system, usually because a system property exhibits a value outside of expected bounds, or has a degrading trend.
3. **Plan:** is concerned with determining a course of action to adapt the managed element once a problem is detected.
4. **Execute:** is concerned with carrying out a chosen course of action and effecting the changes in the system.

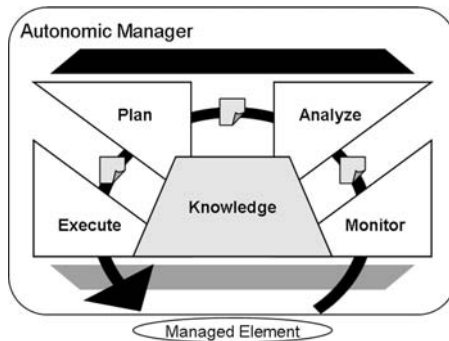


Fig. 1 The IBM Autonomic MAPE Reference Model

Shared between these four phases is the **Knowledge** component, which contains models, data, and plans or scripts to enable separation of adaptation responsibilities and coordination of adaptations. The Rainbow framework provides components that fulfill each of these four phases and the *knowledge* to support self-adaptation.

3 Software Architecture and Architecture-Based Self-Adaptation

A key issue in using an external model is to determine the appropriate kind of models to use for software-based systems. Each type of model has certain advantages in terms of the analyses and kinds of adaptation it supports. In principle, a model should be abstract enough to allow straightforward detection of problems in the target system, but should provide enough fidelity to determine remedial actions to take to fix the problem. State machines, queuing theory, graph theory, differential equations, and other mathematical models [40, 42] have all been used for model-based, external adaptation of software systems.

We, among others, use a system's software architecture as the external model for dynamic adaptation [19, 39]. The architecture of a software system is an abstract representation of the system as a composition of computational elements and their interconnections [44]. Specifically, an *architecture model* represents the system as a graph of interacting components.¹ Nodes in the graph, termed *components*, represent the principal computational elements and data stores of the system: clients, servers, databases, user interfaces, etc. Arcs, termed *connectors*, represent the pathways of interaction between the components. This is the core architectural representation scheme adopted by a number of architecture description languages (ADLs), such as Acme [20] and xADL [13].

The use of software architecture as the basis for self-adaptation, termed *architecture-based self-adaptation*, holds a number of potential promises. A rich body of work on architecture trade-off analysis techniques used at system design time facilitates runtime self-adaptation. As an abstract model, an architecture model provides a global perspective on the system and exposes the important system-level behaviors and properties. As a locus of high-level system design decisions, the model makes system integrity constraints explicit, thereby helping to ensure the validity of a change. For example, the architecture model can expose important properties such as throughput and bandwidth, allowing the overall throughput or performance of the system to be analyzed. Furthermore, the model might be associated with explicit constraints on the architecture that, for example, forbid cycles. This knowledge can be used at runtime to reason about the effect of a change on the system's throughput or structure. See [18] for a discussion of this concept for performance evaluation.

¹ We are primarily interested in the component–connector view [11] because it characterizes the abstract state and behavior of the system at runtime to enable reasoning about problems and courses of adaptation.

Crucial for architecture-based self-adaptation is the choice of the *architectural style* used to represent the target system. A style (e.g., pipe-filter) provides the vocabulary to describe the architecture of a system in terms of a set of component types (e.g., filter) and connector types (e.g., pipe), along with the rules for composition (e.g., no cycles) [1]. A style might also prescribe the properties associated with particular element types (e.g., throughput on a pipe). Usually associated with a style is a set of analytical methods to reason about properties of systems in that style. For example, systems in the MetaH style supports real-time schedulability analysis [16].

For self-adaptation, given some quality objectives, each style may guide the choice of system properties to monitor, help identify strategic points for system observation, and suggest possible adaptations. To illustrate this, consider a signal-processing system with an architecture in the pipe-filter style. This style constrains the system to a data-flow computation pattern, points to throughput as a system property, identifies the filter as a strategic point for measuring throughput, and suggests throughput analysis for reasoning about overall system throughput. The pipe-filter style may suggest adaptations that swap in variants of filters to adjust throughput, create redundant paths to improve reliability, or add encryption to enhance security. In contrast, consider a different system in the client-server style. This style highlights request-response latency as a key property, identifies the client as a strategic point for measuring latency and the server for load, and suggests the use of queuing theory to reason about service time and latency. The style may suggest an adaptation that switches clients to less loaded servers to reduce latency.

4 Related Work

To date, several dynamic software architectures and architecture-based adaptation frameworks have been proposed and developed [7, 24, 39], including an effort to characterize the style requirements of *self-healing systems* [35]. Below, we examine a representative set of approaches, categorizing each by its primary focus, then highlighting its main features. Broadly speaking, related approaches focus on formalism and modeling, or mechanisms of adaptation. A third category addresses distribution and decentralization of control.

4.1 *Distributed, Decentralized Adaptation*

Work on self-organizing systems in [23] proposes an approach where self-managing units coordinate toward a common model, an architectural structure defined using the architectural formalism of Darwin [33]. Each self-organizing component is responsible for managing its own adaptation with respect to the overall system. To do this, each component maintains a copy of the architecture model of the entire system. While this approach provides the advantage of distributed control and eliminates a single point of failure, requiring each component to maintain a global model

and keep the model consistent, which imposes significant performance overhead. Furthermore, the approach prescribes a fixed distributed algorithm for global configuration. We overcome the performance overhead and coordination issue by allowing tailorable global reorganization without imposing a high-performance overhead, but we trade off distributed, localized control of adaptation decision.

4.2 Formal, Dynamic Architectures

A number of approaches focus on modeling and formalizing dynamic systems, rather than mechanisms to enable self-adaptation. Our approach builds on formal architectural modeling, using the model within a framework of reusable infrastructures to enable self-adaptation in a target system. Wermelinger and colleagues developed a high-level language, based on CommUnity, to describe architectures, as well as changes over an architectural configuration, such as adding, removing, or substituting components or interconnections [49].

The K-Component model addresses the integrity and safety of dynamic software evolution, modeled as graph transformations of meta-models on architecture [15]. It uses reflective programs called adaptation contracts to build adaptive applications, coordinated via a configuration manager (similar to Le Métayer's approach [31]).

Darwin is an ADL for specifying the architecture of a distributed system, with an operational semantics that captures dynamic structures as the elaboration of components and their bindings in a configuration [33]. Organization of components and connectors may change during execution. The evolving structures of Darwin are modeled using Milner's π -calculus, allowing the correctness of its program elaboration to be analyzed. Together with its π -calculus semantics, Darwin serves as a general-purpose configuration language for specifying distributed systems. ArchWare [37] and PiLar [12] are examples of ADLs that use architectural reflection to model layers of active architectures, allowing separate concerns to be addressed at different layers. These approaches rely on sophisticated reflective technologies to support the active architectures and enable dynamic co-evolution.

These approaches assume that system implementations are generated from the architecture descriptions. In contrast, our approach relies on external mechanisms decoupled from the target system and can therefore be used to add adaptation to existing systems.

4.3 Style-Specific Approaches with Fixed Quality Attributes

A number of architecture-based approaches provide mechanisms to enable self-adaptation (or system reconfiguration) that focus on particular quality attributes of systems, such as performance [6, 27, 32], survivability [50], or that focus on particular architectural styles, for example, [26, 38].

Most closely related to our own work is that of the UCI Research group headed by Taylor [14], and the research of Sztajnberg [47]. As a natural extension of [38], Taylor's group developed an architecture-based runtime architecture evolution framework, which dynamically evolves systems using a monitoring and execution loop controlled by a planning loop. This framework supports self-adaptation for C2-style systems, and evolution of the architecture model uses architectural differencing and merging techniques similar to those used for source code version control. Sztajnberg and Loques developed the CR-RIO framework, which uses a style-neutral ADL (CBabel), architectural contracts to specify execution context, application profiles to describe resource requirements, and middleware to perform architectural reconfigurations based on the specified contracts. CR-RIO demonstrates a formal verification capability but does not appear to support automation of multi-objective adaptations, for example by composing multiple contracts, nor does it address engineering aspects. Our approach can be applied to different classes of systems and can address multiple quality objectives.

Current approaches present a number of limitations and unresolved issues, which are addressed by Rainbow. In particular, where traditional adaptive techniques—for example, the ones based on exception-handling mechanisms and network timeouts—rely only on localized knowledge of system states, we use an architecture-based approach to leverage a more global perspective. While existing approaches do not address the quantity of adaptation and system-level details that engineers grapple with in order to build self-adaptation for their systems, we design a language that encapsulates core self-adaptation concepts and hoists them as first-class building blocks for system engineers to build self-adaptation capabilities. Finally, almost no existing approach provides a systematic, integrated approach to self-adaptation that combines an end-to-end system perspective, style-based adaptation, automation of routine human expertise, and incremental support to developing self-adaptation capabilities; we address this by providing a framework with reusable infrastructures and customizable elements.

5 The Rainbow Approach

Related work provides some of the building blocks for our own research. Software architecture research provides the language, models, and analysis mechanisms to represent and reason about a system's runtime properties; related work in self-healing systems and architecture-based approaches demonstrate the effectiveness of using software architecture for particular classes of systems and fixed quality attributes. What is missing is an approach to self-adaptation that (a) is generally applicable to different classes of systems and quality objectives, (b) allows adaptation to be represented as explicit operational entities and chooses the best one in a principled and analyzable way, and (c) provides an integrated approach that saves engineers time and effort in writing and changing adaptation.

Our approach satisfies the above requirements by (1) providing a framework, called Rainbow, that provides general, supporting mechanisms for self-adaptation,

and which can be tailored to different classes of systems and (2) defining a language, called “Stitch,” that plugs into this framework and allows adaptation expertise to be specified and reasoned about, and which can be used to automate and coordinate adaptations to satisfy multiple objectives.

The Rainbow framework is illustrated in Fig. 2. It functions as follows. Monitoring mechanisms—*probes* and *gauges*—observe the running *target system*. Observations are reported to update properties of the architecture model managed by the *Model Manager*. The *Architecture Evaluator* evaluates the model upon update to ensure that the system is operating within an acceptable range, as determined by architectural constraints. If the evaluation determines that the system has a problem, the Evaluator triggers the *Adaptation Manager* to initiate the adaptation process and choose an appropriate repair strategy. The *Strategy Executor* executes the strategy on the running system via system-level *effectors*.

There are three important components to making our solution work: (1) software architecture gives us leverage to make self adaptation general and cost-effective; (2) control theory provides a well understood mechanism for closed-loop system adaptation; and (3) utility theory allows us to pick the most appropriate strategy for repair. Details of each of these are enumerated below.

5.1 The Elements of Rainbow

5.1.1 Software Architecture Model and Style

The first major element of Rainbow is the use of a *stylized* software architecture model to monitor and adapt a target system. Like the blueprint of a building, the

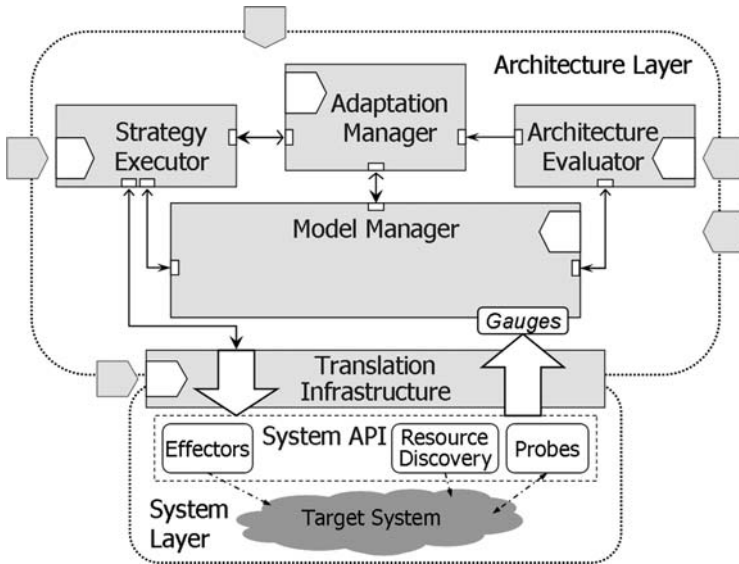


Fig. 2 The Rainbow framework with notional customization points

software architecture model of a system provides an abstract view of the modeled software system. The architecture model elides low-level details and allows the architect to focus on the important, high-level properties of the system. The model is described using a particular vocabulary that conveys the structural characteristics of the system, for example, client–server, dataflow, N-tier, and repository. Current approaches to architecture modeling also allow the architect to specify explicit rules, or constraints, about element composition in the system. An architecture model so specified enables the architect to analyze the system for quality attributes such as performance, availability, reliability, and security. Together, *vocabulary*, *rules*, *properties*, and *analyses*, summarized below, comprise the building blocks of *architectural style* [1, 44].

1. **Vocabulary** (*V*) of element types, including component types (e.g., database, client, server, filter), connector types (e.g., sql, http, rpc, pipe,), and component and connector interface types.
2. **Design rules** (*R*), or constraints, that determine the permitted composition of those elements. For example, the rules might require every client in a client–server organization to connect to at most one server, prohibit cycles in a particular pipe-filter style, or define a compositional pattern such as a starfish arrangement of a blackboard system or a pipelined decomposition of a compiler.
3. **Properties** (*P*) that are characteristic of elements in a style, in particular to provide analytic and sometimes behavioral or semantic information. For instance, “load” and “service time” properties might be characteristic of server elements in a performance-specific client–server style, while “transfer-rate” might be a common property in a pipe element of a pipe-filter style.
4. **Analyses** (*A*) that can be performed on systems built in that style. Examples include performance analysis using queuing theory for a client–server system [46] and schedulability analysis for a style oriented toward real-time processing [3].

While this traditional notion of style suffices to model snapshots of a system’s architecture, including dynamic behavior of, and interactions between, system elements (e.g., Darwin [33] and Wright [4]), this characterization of style lacks mechanisms to explicitly represent what dynamic architectural changes are allowed by systems of the style. Capturing allowable operations to the system is important for modeling, analyzing, and reasoning about dynamic system adaptation. For example, knowing whether a system’s style allows the activation of a server or the swap of a communication channel helps determine possible adaptations for that system.

To handle the notion of dynamism with respect to architectural structure, we augment the notion of style with *operators*.

5. **Operators** (*O*). A set of style-specific operations that may be performed on elements of a system to alter its configuration. For example, a service-coalition style might define operators `addService` or `removeService` to add or remove a service from a system configuration in this style.

The notion of *architectural style* (augmented with *operators*) gives the architect a powerful abstraction to describe, classify, and analyze many different kinds of systems. Style provides the unifying concepts to factor commonalities out of classes of system and to characterize differences between those classes. Specifically, we leverage style in our design of the Rainbow approach and framework, in combination with the runtime use of architecture and environment models, to achieve generality and cost-effectiveness. We present its design and customization points in Sect. 5.3. Next, we discuss control systems theory, which is integral to the design of our self-adaptation framework.

5.1.2 Control Systems and the Self-Adaptation Cycle

The second major element of Rainbow is the application of control systems concepts to the adaptation problem. Self-adaptation requires a closed loop of control. We choose a specific type of control system model to make our approach generalizable and reusable across different classes of system. In a typical control system, the Controller must have access to relevant *Measured Output* from the target system as well as maintain control over some *Control Input*. In our context, the target system is the software system that requires self-adaptation. Controlling a software system requires mechanisms to obtain information about the system and its execution environment. Therefore, in addition to maintaining a model of the system's architecture, some model of the system's execution environment must also be maintained. Also, the Controller must be able to select a course of action and effect changes on the system.

These required capabilities of control correspond to the 4 + 1 phases of the adaptation cycle defined by the IBM Autonomic MAPE Architecture mentioned in Sect. 2.1 [17]: *knowledge* is embodied in the architecture model, managed by the Model Manager, *monitoring* is achieved by Probes and Gauges updating the model, *detection* is performed by the Architecture Evaluator assessing problems on the model, *decision* occurs through the Adaptation Manager choosing a remedy based on model states, and *action* is accomplished by the Strategy Executor effecting changes on the system via Effectors. For the *decision* phase, in order to represent and reason about the courses of remedy, we introduce *strategy* as a concept of self-adaptation. Each adaptation decision requires the consideration of multiple factors, which leads to the third element used by Rainbow: utility theory.

5.1.3 Utility Theory

Once a problem is detected by Rainbow, an appropriate adaptation must be chosen. To be effective, such a choice must consider overall business objectives and priorities, and decide between multiple potential adaptations that have possible interacting effects on the system (e.g., an adaptation that fixes performance might affect security concerns, and vice versa). To deal with this, our approach uses utility theory.

To determine the most appropriate strategy in a given circumstance, we need to define values for the objectives, relate the objectives to specific system conditions,

and assess the impact of the strategies on the objectives. One important concern is the uncertainty in the outcome of a particular strategy: enacting a strategy does not necessarily mean that the strategy will be successful on the system. This uncertainty is due to a number of factors, including intervening operation of the system between problem detection and adaptation, inadequate knowledge of the environment, or unanticipated errors in strategy execution. We address this by combining utility theory with a stochastic model of the strategy outcomes. This provides a method to quantify strategies relative to the objectives, under uncertainty.

5.2 Znn.com Example

To illustrate the framework, consider an example news service, *Znn.com*, that serves multimedia news content to its customers, inspired by real sites like *cnn.com* and *rockymountainnews.com*. Architecturally, *Znn.com* is a web-based client-server system that conforms to an N-tier style. As illustrated in Fig. 3, *Znn.com* uses a load balancer to balance requests across a pool of replicated servers, the size of which is dynamically adjusted to balance server utilization against service response time. A set of client processes (represented by the *C* component) makes stateless content requests to one of the servers. Let us assume we can monitor the system for information such as server load and the bandwidth of server-client connections. Assume further that we can modify the system, for instance, to add more servers to the pool or to change the quality of the content. We want to add self-adaptation capabilities that will consider monitored information and adapt the system to fulfill *Znn.com* objectives.

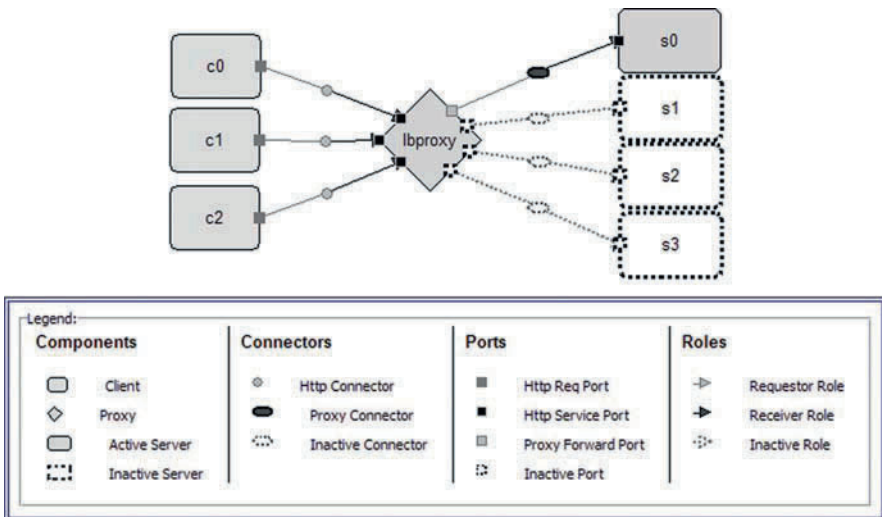


Fig. 3 Architecture model of the Znn.com system

The business objectives at Znn.com require that the system serve news content to its customers within a reasonable response time range while keeping the cost of the server pool within its operating budget. From time to time, due to highly popular events, Znn.com experiences spikes in news requests that it cannot serve adequately, even at maximum pool size. To prevent unacceptable latencies, Znn.com opts to serve only textual content during such peak times in lieu of providing its customers zero service. The Znn.com system administrators (sys-admins) adapt the system using two actions: adjust the server pool size or switch content mode. When the system comes under high load, the sys-admins may increase the server pool size until a cost-determined maximum is reached, at which point the sys-admin switches the servers to serve textual content. If the system load drops, the sys-admin may switch the servers back to multimedia mode to make customers happy, in combination with reducing the pool size to reduce operating cost.

The adaptation decision is determined by observations of overall average response time versus server load. Specifically, four adaptations are possible, and the choice depends both on the conditions of the system and on business objectives:

1. Switch the server content mode from multimedia to textual
2. Switch the server content mode from textual to multimedia
3. Increment the server pool size, and
4. Decrement the server pool size

We want to help Znn.com automate system management to adjust the server pool size or to switch content between multimedia and textual modes. In reality, a news site like *cnn.com* already supports some level of automated adaptation. However, automating decisions that trade off multiple objectives to adapt a system is still unsupported in most systems today. For instance, while automating adaptations on performance concerns is possible (e.g., load balancing), it is much harder to do so in the presence of conflicting qualities such as security.

In terms of Znn.com, the average response time and server load for Znn.com are monitored and those measurements update corresponding properties in the Znn.com architecture model managed by the Znn.com-customized Model Manager. The customized Architecture Evaluator evaluates the model to make sure that no client experiences a request-response latency above a certain threshold. If a client is experiencing above-threshold latencies, the Evaluator triggers the Adaptation Manager to initiate the adaptation process and determine whether to activate more servers or decrease content quality. The customized Strategy Executor carries out the strategy on the Znn.com system using the provided system hooks.

Building a self-adaptive system such as that outlined above is a costly proposition if the important components such as the monitoring, model management, adaptation, and translation mechanisms have to be built from scratch. For this reason, we have engineered an integrated framework with shared infrastructures and developed an iterative process to facilitate reuse of self-adaptive functionalities and reduce the cost and effort of achieving self-adaptation.

5.3 Tailorable Rainbow Framework

Rainbow is a framework with general and reusable infrastructure services that can be tailored to particular system styles and quality objectives, and further customized to specific systems. The customization is notionally illustrated as plug-in pieces in Fig. 2. The *Rainbow framework* consists of a number of components that provide the monitoring, detection, decision, and action capabilities of self-adaptation.

This customizable self-adaptation framework has a number of advantages. Providing a substantial base of reusable infrastructure greatly reduces the cost of development. Providing separate customization mechanisms allows engineers to tailor the framework to different systems with relatively small increments of effort. In particular, the tailorable model management and adaptation mechanisms give engineers the ability to customize adaptation to address different properties and quality concerns, and to add and evolve adaptation capabilities with ease. Furthermore, a modular adaptation language to specify the adaptation policy allows engineers to consider adaptation concerns separately and then compose them.

5.3.1 Rainbow Models

The Rainbow framework leverages two kinds of models to make adaptation decisions: the architecture model and the environment model. An architecture model reflects abstract, runtime states of the target system itself. Many current approaches do not consider the system context, or environment, to make adaptation decisions. Rainbow addresses this shortcoming through an explicit treatment of *environment* states in the self-adaptation process. An environment model provides contextual information about the system, including the executing environment and its resources. For example, if additional servers are needed, the environment model indicates what spare servers are available. When a better connection is required, the environment model has information about available bandwidth on other communication paths.

Managing an executing system dynamically requires knowing the entities that are present, the runtime states they are in, and how they communicate. As noted, the architecture model captures the state of the system as a graph of interacting, communicating entities representing the Component and Connector (C&C) view of architecture [11]. It consists of an instance of the target system defined in a particular style, associated properties and their dynamically updated values, and constraints on the structure of the target system.

The architecture of the system for the Znn.com example is described in the ClientServerFam style with component, connector, and property types for clients, servers, and HTTP connections. Clients in this system define an *average_latency* property value and an architectural constraint specifying that this property should always be below a threshold.

The environment model captures states of the target system's execution environment to provide additional information for the self-adaptation process. Information about the various *resources* must be sufficient to facilitate reasoning about adapta-

tion. As with architecture, we represent environment information as a graph where nodes represent resources and typed edges represent relations between resources, such as physical connection, containment, and dependencies. We capture common relation and resource types in an *environment style*. Environment resources typically relate closely with system elements, so we maintain a mapping between architecture-model elements and environment-model elements.

5.3.2 Translation Infrastructure—Monitoring and Action

In order to get information out of the target system into an abstract model for management, and then to push changes back into the system, the layer marked *Translation Infrastructure* in Fig. 2 provides **monitoring** and **action** (cf., Sect. 5.1.2) hooks, and bridges the abstraction gap between the system and the architecture model. This infrastructure builds on prior work and encompasses monitoring mechanisms, action mechanisms, and various sets of correspondence mappings [5, 10, 22].

Monitoring Mechanisms: *Probes* and *gauges* extract system states, then aggregate and abstract them to update the model. Intuitively, a probe measures some part of the system, while a gauge interprets that measurement to provide a reading. In Rainbow, as illustrated in Fig. 4, probes are deployed onto the target system to measure and publish system information, such as CPU load or process run state. Gauges are associated with specific properties in the architecture model; they collect, aggregate, and abstract probe measurements to populate corresponding architectural properties. Different kinds of probes are deployed onto the target system to detect system states (e.g., whether compression across a communication link is enabled), measure quality attributes (e.g., link latency or intrusion detector state), and discover resources (e.g., to find an available Apache server). Likewise, different types of gauges are needed to aggregate and interpret system properties (e.g., to average latency).

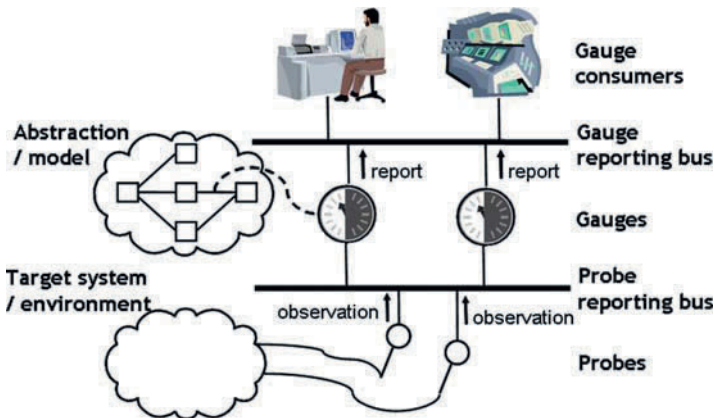


Fig. 4 Monitoring mechanisms: probes and gauges

To tailor the monitoring mechanisms, an adaptation engineer identifies the properties of specific element types to monitor and finds matching gauges and probes from gauge and probe libraries to monitor those properties (or develops them if none are available). The engineer maps the gauge-updated property to the architectural property via the *mapping* attribute, and also defines the target probe, by type name, to which the gauge maps. While we require probes and gauges to enable overall Rainbow functionality, they are not the focus of this chapter.

Action Mechanisms: *Effectors* carry out change operations on the target system; they are associated with architectural operators in the Rainbow Architecture Layer (Fig. 2). Under the hood, the mechanism to realize an effector could range in complexity from a system-call, to a script, to a complex, workflow-based subsystem (e.g., KX Worklets [48]). As with probes and gauges, we require effectors to enable overall Rainbow functionality, but they are not the focus of this chapter.

Rainbow’s dependency on monitoring and action capabilities for the target system is not a serious limitation. We build on other researchers’ work on probing and effecting capabilities, including adaptive middleware technology [2, 8]. Furthermore, modern systems increasingly support probing and effecting functionalities, as evidenced by products from industry initiatives such as IBM’s Autonomic Computing [17] and Microsoft’s Dynamic Systems Initiatives [34].

Translation Mappings: Our use of an abstract model to monitor and control the target system requires us to bridge the abstraction gap with correspondence mappings. In a prior publication [10], we identified four distinct kinds of correspondence mappings, maintained by the Translation Infrastructure, to facilitate translation of control information between the architecture model and the target system. For example, when the Strategy Executor invokes an effector, arguments to be passed to the effector must be translated from architectural elements to target-system entities. We briefly summarize the mappings below:

- A **Type** map relates a type of element in the architecture model with a type of entity in the target system, including any properties defined for the type of element/entity.
- An **Element** map relates an element instance in the architecture model with an entity in the target system, including the property values.
- An **Operation** map relates an architectural operator, along with its formal parameters (type and name), to an effector with its corresponding parameters.
- An **Error** map relates the identifier and error sources of an exception in the target system to a corresponding error at the architecture level.

5.3.3 Model Manager

The *Model Manager* manages both the architecture and environment models of the target system. It maintains references between elements of the environment and the architecture models. It tracks the model states, maintains correspondence between the model and the system and environment states via gauges, provides the Rainbow components with shared access to the models via query and modify APIs, and

deploys gauges (and corresponding probes) as dictated by model property queries. Elements in both the architecture and the environment models are accessed via direct model reference in the adaptation scripts (e.g., `EnvModel.elementX.prop`).

To tailor the Model Manager, it is sufficient to tailor the managed models. A style writer specifies a vocabulary (a family of element types) to describe the architecture of the target system, defines the architecture and environment model instances, and identifies the relevant properties to collect via the monitoring infrastructure.

5.3.4 Architecture Evaluator

Armed with a model that captures runtime system and environment states, we need a mechanism to **detect** when an adaptation is needed (cf., Sect. 5.1.2). When any model property changes, the *Architecture Evaluator* evaluates the conformance of the architecture model to a predefined set of constraints. Upon detecting a constraint violation, it notifies the Adaptation Manager (Fig. 2) to trigger adaptation. This mechanism leverages prior work on the use of architectural constraints, specified in first-order predicate logic, to identify flaws in system design [36]. We extend this work by checking architectural constraints over runtime system properties to detect target system problems at runtime.

To tailor the Evaluator, a style writer specifies as rules the topological and behavioral constraints that (a) characterize the bounds of the target system and/or (b) signify opportunities for adaptation. These architectural rules are specified in the architecture model as first-order predicate logic expressions over architectural structure and properties.

5.3.5 Adaptation Manager

Once a problem is detected, we need a mechanism to **decide** on the appropriate adaptation remedy (cf., Sect. 5.1.2). When triggered by the Architecture Evaluator, the *Adaptation Manager* uses the architecture model to select a remediation strategy that best suits the present problem state of the system, then coordinates the execution of that strategy. Automating system adaptation requires formalizing three kinds of information to instruct the machine to act automatically: for *what* to adapt, *when* to adapt, and *how* to adapt the system.

A quality dimension determines *what* to adapt for and corresponds to a business quality of concern, which is characterized as a utility function and mapped to a monitored architectural property. For example, Average response time (uR) is mapped to `ClientT.experRespTime` in the architecture and has the utility function defined by the points $\langle (0, 1), (500, 0.9), (1500, 0.5), (4000, 0) \rangle$ to represent the utility of average response time at 0, 500, 1500, and 4000 ms. The utility of values of points in between are interpolated. To manage multiple objectives, each quality of concern is given a relative weight that captures business preferences across the quality dimensions. To help decide *when* adaptations are applicable we specify conditions of applicability, e.g., `invariant self.avg_latency < MAX_RESPTIME`.

```

1  module newssite.strategies.example;
2  import model "ZnnSys.acme" { ZnnSys as M, ZnnFam as T };
3  import lib "newssite.tactics.example";
4  import op "org.sa.rainbow.stitch.lib.*"; // Model, Set, & Util
5
6  define boolean styleApplies = ...
7  define boolean cViolation = exists c : T.ClientT in M.components |
8      c.experRespTime > M.MAX_RESPTIME;
9
10 strategy SimpleReduceResponseTime [ styleApplies && cViolation ] {
11     define boolean hiLatency = ...
12     define boolean hiLoad = ...
13
14     t1: (hiLatency) -> switchToTextualMode() {
15         t1a: (success) -> done ; }
16     t2: (hiLoad) -> enlistServer(1) {
17         t2a: (!hiLoad) -> done ;
18         t2b: (!success) -> do [1] t1 ; }
19     t3: (default) -> fail;
20 }

```

Fig. 5 An example *strategy* SimpleReduceResponseTime

The Stitch self-adaptation language allows strategies to be specified that capture a pattern of adaptations in which each step evaluates a set of condition-action pairs and executes an action, possibly waiting for the action to take effect. Actions use operators on the architectural style to make changes to the system. A strategy also specifies conditions of applicability that determine in what contexts it should be involved. Furthermore, we need to specify cost–benefit attributes to relate its impact on the quality dimensions. Detailed language features appear in [9].

The adaptation process works as follows: When the *Architecture Evaluator* detects an *adaptation condition*, it triggers the *Adaptation Manager* to initiate a round of adaptation. The Adaptation Manager first checks the strategy *conditions of applicability* to filter a subset of applicable *strategies* based on current system conditions (reflected in the model). In Fig. 5, SimpleReduceResponseTime applies when the conditions styleApplies (definition elided in line 6) and cViolation (defined in lines 7 and 8) are true. The Adaptation Manager then selects the best strategy from the subset by computing the expected utility of each strategy. Briefly, the expected utility of each strategy is computed by first computing the expected aggregate impact of each strategy on each *quality dimension* using the specified *cost–benefit attributes*. Next, the strategies are scored using the *utility preferences* over the quality dimensions. Finally, the highest scoring strategy is selected.

The Adaptation Manager combines utility, decision, and control theories to solve the decision-making problem in self-adaptive systems. To tailor the Adaptation Manager, the engineer specifies a set of adaptation strategies, the quality dimensions and utility preferences, and the cost–benefit attributes to enable automated selection of strategies.

5.3.6 Strategy Executor

Once a strategy is chosen, we need a mechanism that can carry out the adaptation on the target system. The *Strategy Executor* is dispatched by the Adaptation Manager to do this. It resolves model references within the strategy against the Rainbow model, observes model states and evaluates branch conditions to determine operators to execute and corresponding system-level effectors to carry out changes.

The Strategy Executor is tailored by the set of operators of the style. For example, for Znn.com, operators would include addServer, removeServer, and setFidelity.

5.4 Rainbow Application to Znn.com

To illustrate how to customize the Rainbow framework, let us walk through the Znn.com example. Table 1 gives an overview of how each of the Rainbow components is customized for Znn.com. This example is simplified to illustrate only the major features of Rainbow.

The stakeholders in the Znn.com example are the customers and the news service provider. The customers care about quick response time of their news requests and high content quality (i.e., multimedia over textual). While aware of the customer content quality preferences, the provider is constrained by infrastructure provisioning costs to provide the service. We use these three quality concerns to define the quality dimensions, which correspond to measurable properties in the target system. We capture each dimension as a discrete set of values:

1. Response time: low, medium, high
2. Quality: graphical or multimedia
3. Budget: within or over

We elicit from the service providers the utility values and preferences for these dimensions. In addition, since response time is affected by the amount of time required to complete an adaptation, we also need to consider a fourth dimension, disruption, which should be minimized. We use an ordinal scale of 1–5 to express the degree of disruption. Cost–benefit attributes necessary for strategy selection are

Table 1 Znn.com: example application of the Rainbow framework

Set	Rainbow component	Customization content highlight
<i>Objective</i>	Adaptation Manager	timely response (uR), high-quality content (uF), low-provisioning cost (uC)
Vocabulary	Model Mgr, Translators	ClientT, ServerT, DatabaseT, HttpConnT
Property	Architecture Evaluator, Monitoring Mechanisms	ClientT.reqRespLatency, HttpConnT.bandwidth, ServerT.load, ServerT.fidelity, ServerT.cost
Rule	Architecture Evaluator	ClientT.reqRespLatency \leq MAX.LATENCY
Operator	Strategy Executor	addServer, removeServer, setFidelity
Strategy	Adaptation Manager	SwitchToTextualMode, SwitchToMultimediaMode, EnlargeServerPool, ShrinkServerPool

Table 2 Znn.com quality dimensions and utility preferences

Label	Description	Architectural property	Utility function	Weight
u_R	Avg Response Time	ClientT.experRespTime	$\langle (low, 1), (med, 0.5), (high, 0) \rangle$	0.4
u_F	Avg Content Quality	ServerT.fidelity	$\langle (textual, 0), (multimedia, 1) \rangle$	0.2
u_C	Avg Budget	ServerT.cost	$\langle (within, 1), (over, 0) \rangle$	0.3
u_D	Disruption	ServerT.rejectedRequests	$\langle (1, 1), (2, 0.75), (3, 0.5), (4, 0.25), (5, 0) \rangle$	0.1

specified with respect to these four quality dimensions. Given our understanding of the quality dimensions, we can specify discrete utility functions for these four dimensions and complete the utility profiles. To determine the utility preferences, assume that Znn.com considers response time the most important, followed by budget, then content quality, and finally disruption. The quality dimensions and utility preferences are summarized in Table 2.

As part of the N-tier style of Znn.com, a set of element types are defined to model elements of the system architecture: ClientT to model client instances, ServerT for server instances, DatabaseT for databases in the data layer, and HttpConnT as one of the prominent protocols of communication. Properties corresponding to the objectives are defined on the style elements to help measure and assess satisfaction of the objectives; respectively, they are ClientT.reqRespLatency, ServerT.fidelity, ServerT.cost, shown in Table 2. These and other properties are measured by probes and gauges in the translation infrastructure.

A rule specifies the acceptable bound of request-response latencies experienced by a client: exceeding MAX_LATENCY indicates a problem. A set of operators correspond to available effectors in Znn.com: the system can be controlled to add or remove servers, or to change the fidelity of the served content.

When Rainbow is customized as above, during operation the Model Manager deploys gauges and corresponding probes on Znn.com to monitor server status, connection bandwidths, and request-response latencies experienced by the clients (can be approximated via server-side proxy). Probes usually report instantaneous and low-level values, while gauges aggregate and average these measurements and report them as values of corresponding architectural properties to the Model Manager. When the Model Manager updates the architecture model, the Architecture Evaluator checks the model to make sure that the constraint is satisfied, i.e., no client experiences a request-response latency above the maximum threshold.

If a client experiences above-threshold latencies, a constraint violation occurs, and the Evaluator triggers the Adaptation Manager to initiate adaptation. The Adaptation Manager scans through a repertoire of strategies, filtering out the inapplicable ones, then scores them to determine expected utility.

The Znn.com example has four possible strategies, corresponding to each of the adaptations outlined in Sect. 5.2: SwitchToTextualMode, SwitchToMultimediaMode, EnlargeServerPool, and ShrinkServerPool. We also specify cost-benefit attribute vectors for these strategies, not shown here, that relate the impact of each

Table 3 Znn.com strategies and cost–benefit impact

Strategy	u_R	u_F	u_C	u_D	Utility
SwitchToTextualMode	-2⇒low	-1 ⇒textual	+0⇒within	3	0.75
EnlargeServerPool	-2⇒low	+0⇒multimedia	+1⇒over	1	0.70

strategy to the four quality dimensions. For example, SwitchToTextualMode lowers the response time and the fidelity level, does not affect the cost, and incurs some level of disruption.

Let us assume that Znn.com hits a peak load period, and the system state falls into a problem state in which the response time is high, the infrastructure cost is within budget, and the content mode is multimedia. In this case, only the strategies SwitchToTextualMode and EnlargeSeverPool are applicable. So we need to score the strategies to determine which one to choose given the utility preferences. The cost–benefit attribute vectors would yield aggregate attribute vectors and utility scores for the two strategies as shown in Table 3.

The utility scores indicate DropFidelityStrategy as the better adaptation strategy, given the current system conditions. The Adaptation Manager delegates the execution of this chosen strategy to the Strategy Executor, which evaluates the strategy and invokes the setFidelity operator. This operator is mapped to a corresponding effector to change the Znn.com system. Once changes are effected, Rainbow’s adaptation cycle continues to monitor system states.

Note that if Znn.com attributed a lower weight to budget, or a higher weight to disruption, or swapped the importance of disruption versus budget, then the other strategy would have scored higher. Using such utility-based analysis, we can choose a strategy by considering four dimensions and accounting for trade-offs across those using the additional input of business utility preferences.

6 Conclusions and Ongoing Work

In this chapter, we described our approach to architecture-based self-adaptation, which allows engineers to add self-adaptation facilities to existing systems. This approach, called Rainbow, involves adding an external mechanism to monitor and enact changes in systems. We summarized the elements of Rainbow, and how they can be customized to different styles of systems and quality dimensions of interest. Our approach is two-pronged: we provide a framework of reusable infrastructure that can be tailored to particular domains and we provide a language called Stitch that can allow adaptation techniques to be codified. We have given an intuition behind the approach as applied to a simple networked system. Interested readers are referred to [9] for details of the customization and the Stitch language.

As summarized in Table 4, we applied Rainbow to a number of systems, including two small client-server systems (CSSys and UnivSys), a service-coalition system (Libra), and two N-tier systems (Znn.com and the infrastructure of Talk-Shoe.com). In all cases we applied Rainbow to adapt the target system within some

Table 4 Summary of applying the Rainbow approach

Claim	CSSys	Libra	UnivSys	TalkShoe	Znn.com	SysAdm	Netbwe
General—Rainbow applies to many <i>styles</i> and multiple <i>objectives</i> ?							
– 3+ styles	(CS)	(SvcC)	(CS)	(N-tier)	(N-tier)	(SvcC)	(SvcC)
– 3+ objectives	(<i>perf</i>)	(<i>perf+cost</i>)	(<i>security</i>)	(<i>avail.</i>)	(4)	(<i>bw+avail.</i>)	(3)
Cost-effective—Rainbow demonstrates <i>reuse</i> (between instances) and <i>ease of use</i> ?							
– Reusable	✓		×	✓		×	×
– Easy to use	×	×	×	✓ 93h	✓ 34h	×	×

×: not applicable, not demonstrated.

user-specified quality goal (e.g., availability), and in the case of Znn.com, we also demonstrated that Rainbow self-adaptation achieved multiple objectives [9]. Finally, we demonstrated that Rainbow could codify system administration tasks.

Our experience with using this approach on a number of systems has pointed to several open areas of future research:

6.1 Improving Detection and Resolution Capabilities

Our approach favors simple but straightforward detection for rapid recognition of problems using a few key variables, for arguably, greater efficiency and effectiveness [25]. Our approach pushes observations into the model and adaptation is triggered when architectural constraints fail. Alternatively, we would like to explore using more sophisticated quality-of-service (QoS) analyzers to continuously evaluate the system for opportunities of improvement based on QoSs.

Our current utility-based approach considers only information about the current state of the system to choose strategies for adaptation. We have implemented mechanisms that consider some simple historical information to avoid repeating bad actions. We would like to take advantage of more historical information and effects, for example, using machine learning as part of the selection process to avoid oscillation and to improve selection quality. We could also integrate learned predictions to anticipate certain QoS problems, such as an anticipated rise in CPU load, drop in available bandwidth, or even a change in the state of user tasks [41].

6.2 Analyzing Adaptation

One natural question that follows from our approach is how to systematically analyze the behavior of the adaptive system and assure certain system properties? Specifically, we would like to develop analyses that answer the following questions:

- Is an adaptation operation consistent with the architectural style? The challenge is to determine the interaction between structure and behavior in an architectural change. This may be addressed, for example, by Kim’s work [29].

- We have not addressed the issue of asynchrony in automated system self-adaptation, i.e., the effects of an adaptation takes time to propagate into the system, and the Adaptation Manager must take that delay into account when deciding the next step of adaptation. Can we automatically determine the timing delay of an adaptation operation? The challenge is to formalize *effectors* to enable timing analysis.

6.3 Adapting Adaptation

Currently, the utility preference profile and cost-benefit attributes are statically determined. While they can be changed manually, we would like to be able to change these dynamically as user needs change. To do this, we anticipate integrating formal notions of a user's task [21, 45].

Acknowledgments This material is based up work supported by the US Army Research Office (ARO) under grant number DAAD19-02-1-0389 (“Perpetually Available and Secure Information Systems”) to Carnegie Mellon University’s Cylab, and NSF grants IIS0534656 (“Role of Architecture in Facilitating Design Collaboration”) and CNS-0615305 (“Activity-Oriented Computing”). Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any of these funding agencies.

References

1. Gregory D. Abowd, Robert Allen, and David Garlan. Formalizing style to understand descriptions of software architecture. *ACM Trans. Softw. Eng. Methodol.*, 4(4):319–364, 1995.
2. ACM. Adaptive middleware. *Communications of the ACM*, 45(6), June 2002.
3. Robert Allen, Steve Vestal, Dennis Cornhill, and Bruce Lewis. Using an architecture description language for quantitative analysis of real-time systems. In *Proc. of the 3rd International Workshop on Software and Performance*, ACM Press, pages 203–210. 2002.
4. Robert J. Allen. *A Formal Approach to Software Architectures*. PhD thesis, Carnegie Mellon University School of Computer Science, May 1997.
5. Robert Balzer. Probe run-time infrastructure. <http://schafercorp-ballston.com/dasada/2001WinterPI/ProbeRun-TimeInfrastructureDesign.ppt>, 2001.
6. Thaís Vasconcelos Batista, Ackbar Joolia, and Geoff Coulson. Managing dynamic reconfiguration in component-based systems. In *EWSA*, volume 3527 of *LNCS*, Springer, pages 1–17, June 13–14, 2005.
7. Jeremy S. Bradbury, James R. Cordy, Juergen Dingel, and Michel Wermelinger. A survey of self-management in dynamic software architecture specifications. In *WOSS '04: Proc. of the 1st ACM SIGSOFT Workshop on Self-managed Systems*, ACM, New York, pages 28–33, 2004.
8. *Proc. of the Working Conf. on Complex and Dynamic Systems Architecture*, December 12–14, 2001.
9. Shang-Wen Cheng. *Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation*. Technical Report CMU-ISR-08-113, Carnegie Mellon University School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213, May 2008.
10. Shang-Wen Cheng, An-Cheng Huang, David Garlan, Bradley Schmerl, and Peter Steenkiste. An architecture for coordinating multiple self-management systems. In *Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA-4)*, June 2004.

11. Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architecture: Views and Beyond*. Pearson Education, Inc., 2003.
12. Carlos E. Cuesta, Pablo de la Fuente, and Manuel Barrio-Solárzano. Dynamic coordination architecture through the use of reflection. In *SAC '01: Proc. of the 2001 ACM Symposium on Applied Computing*, ACM, New York, pages 134–140, 2001.
13. Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. A highly-extensible, XML-based architecture description language. In *Proceedings of WICSA2*, Massachusetts, USA, August 28–31, 2001. Kluwer Academic Publishers, New York.
14. Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. Towards architecture-based self-healing systems. In Garlan et al. [19], pages 21–26.
15. Jim Dowling and Vinny Cahill. The k-component architecture meta-model for self-adaptive software. In *REFLECTION '01: Proc. of the 3rd International Conf. on Metalevel Architectures and Separation of Crosscutting Concerns*, Springer-Verlag, London, UK, pages 81–88, 2001.
16. Peter H. Feiler, Bruce Lewis, and Steve Vestal. Improving predictability in embedded real-time systems. Technical Report CMU/SEI-2000-SR-011, Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA 15213, December 2000.
17. A. G. Ganak and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.
18. David Garlan, Shang-Wen Cheng, and Bradley Schmerl. Increasing system dependability through architecture-based self-repair. In Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Architecting Dependable Systems*, Lecture Notes in Computer Science, Springer-Verlag, Inc. New York, pages 61–89, 2003.
19. David Garlan, Jeff Kramer, and Alexander Wolf, editors. *Proc. of the 1st ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02)*, ACM Press, New York, November 18–19, 2002.
20. David Garlan, Robert T. Monroe, and David Wile. Acme: Architectural descriptions of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, Cambridge 2000.
21. David Garlan and Bradley Schmerl. The radar architecture for personal cognitive assistance. *International Journal of Software Engineering and Knowledge Engineering*, 17(2), April 2007. A shorter version of this paper appeared in the 2006 Conference on Software Engineering and Knowledge Engineering (SEKE 2006).
22. David Garlan, Bradley Schmerl, and Jichuan Chang. Using gauges for architecture-based monitoring and adaptation. In CDSA [8].
23. Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organizing software architectures for distributed systems. In Garlan et al. [19], pages 33–38.
24. Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing systems - survey and synthesis. *Decis. Support Syst.*, 42(4):2164–2185, 2007.
25. Malcolm Gladwell. *Blink: The Power of Thinking Without Thinking*. Penguin, January 2006.
26. Michael M. Gorlick and Rami R. Razouk. Using Weaves for software construction and analysis. In *Proc. of the 13th International Conf. of Software Engineering*, IEEE Computer Society Press, Los Alamitos, CA, USA, pages 23–34, May 1991.
27. Michael Hinz, Stefan Pietschmann, Matthias Umbach, and Klaus Meissner. Adaptation and distribution of pipeline-based context-aware web architectures. In *WICSA '07: Proc. of the 6th Working IEEE/IFIP Conf. on Software Architecture*, IEEE Computer Society, Washington, DC, page 15, 2007.
28. IBM. An architectural blueprint for autonomic computing, 2004.
29. Jung Soo Kim and David Garlan. Analyzing architectural styles with Alloy. In *Workshop on the Role of Software Architecture for Testing and Analysis 2006 (ROSATEA 2006)*, Portland, ME, July 17, 2006.

30. John C. Knight, Dennis Heimbigner, Alexander L. Wolf, Antonio Carzaniga, Jonathan C. Hill, Premkumar Devanbu, and Michael Gertz. The Willow survivability architecture. In *Proc. of the 4th Information Survivability Workshop*, October 2001.
31. Daniel Le Métayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998.
32. Yan Liu and Ian Gorton. Implementing adaptive performance management in server applications. In *Proc. of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '07)*, IEEE Computer Society, Washington, DC, page 12, 2007.
33. Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In *SIGSOFT '96: Proc. of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, ACM, New York, pages 3–14, 1996.
34. Microsoft Corporation. Dynamic systems initiative. <http://www.microsoft.com/breakwindowsserversystem/dsi/>, 2003.
35. Marija Mikik-Rakic, Nikunj Mehta, and Nenad Medvidovic. Architectural style requirements for self-healing systems. In Garlan et al. [19], pages 49–54.
36. Robert T. Monroe. Capturing software architecture design expertise with Armani. Technical Report CMU-CS-98-163, Carnegie Mellon University School of Computer Science, 1998.
37. Ronald Morrison, Dharini Balasubramaniam, Flávio Oquendo, Brian Warboys, and R. Mark Greenwood. An active architecture approach to dynamic systems co-evolution. In *ECSCA*, volume 4758 of *LNCS*, Springer, New York, pages 2–10. September 24–26, 2007.
38. Peyman Oreizy. *Open Architecture Software: A Flexible Approach to Decentralized Software Evolution*. PhD thesis, University of California, Irvine, 2000.
39. Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May–June 1999.
40. Robert H. Perry, Don W. Green, and James O. Maloney. *Perry's Chemical Engineers' Handbook*. McGraw-Hill, New York, seventh edition, 1997.
41. Vahe Poladian. *Tailoring Configuration to User's Tasks under Uncertainty*. PhD thesis, Carnegie Mellon University School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213, May 2008.
42. Dale E. Seborg, Thomas F. Edgar, and Duncan A. Mellichamp. *Process Dynamics and Control*. Wiley Series in Chemical Engineering. John Wiley & Sons, New York, 1989.
43. Mary Shaw. Beyond objects: A software design paradigm based on process control. *Software Engineering Notes*, 20(1):27–38, January 1995.
44. Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.
45. Joao Pedro Sousa. *Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments*. Technical report cmu-cs-05-123, Carnegie Mellon University School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213, March 28, 2005.
46. Bridget Spitznagel and David Garlan. Architecture-based performance analysis. In *Proc. of the 10th International Conf. on Software Engineering and Knowledge Engineering*, pages 146–151. Knowledge Systems Institute, 1998.
47. Alexandre Sztajnberg and Orlando Loques. Describing and deploying self-adaptive applications. In *Proc. 1st Latin American Autonomic Computing Symposium*, July 14–20, 2006.
48. Giuseppe Valetto, Gail Kaiser, and Gaurav S. Kc. A mobile agent approach to process-based dynamic adaptation of complex software systems. In *8th European Workshop on Software Process Technology*, pages 102–116, June 2001.
49. Michel Wermelinger, Antónia Lopes, and José Luiz Fiadeiro. A graph based architectural (re)configuration language. *SIGSOFT Software Engineering Notes*, 26(5):21–32, 2001.
50. Alexander L. Wolf, Dennis Heimbigner, Antonio Carzaniga, Kenneth M. Anderson, and Nathan Ryan. Achieving survivability of complex and dynamic systems with the Willow framework. In *CDSA* [8].

Mobile Agent Middleware for Autonomic Data Fusion in Wireless Sensor Networks

Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou

Abstract Mobile agents (MAs) are referred to as autonomous application programs with the inherent ability to move from node to node towards a goal completion. In the context of wireless sensor networks (WSNs), MAs may be used by network administrators in the process of combining data and knowledge from different sources aiming at maximizing the useful information content. MAs have been initially developed to replace the client/server model which exhibits many disadvantages, particularly in WSN environments (e.g. heavy bandwidth usage and excessive energy expenditure). The most promising advantages of MAs in WSN environments include decreased usage of the wireless spectrum (large volumes of raw sensory data are filtered at the source) and energy consumption, increased reliability due to their inherent support for disconnected operations, their ability of cloning themselves to enable parallel execution of similar tasks, etc. The main objective of this chapter is to review and evaluate the most representative MA-based middleware proposals for autonomic data fusion tasks in WSNs and evaluate their relevant strengths and shortcomings. Although the chapter's focus is on autonomic data fusion tasks, other applications fields that may benefit from the MAs distributed computing paradigm are identified. Open research issues in this field are also discussed.

1 Introduction

Wireless sensor networks (WSNs) typically comprise hundreds or even thousands of sensor nodes (SNs). These nodes are often randomly deployed in a sensor field and form an infrastructure-less network. Each node has the capability to collect data and route it back to a processing element (PE) or sink via ad hoc connections, using neighbour nodes as relays. A sensor node consists of five basic parts: sensing unit, central processing unit (CPU), storage unit, transceiver unit and power unit [1].

A. Mpitziopoulos (✉)

Department of Cultural Technology and Communication, University of the Aegean Address of Institute, Lesvos, Greece
e-mail: crmaris@aegean.gr

Unlike other type of networks, WSNs are subject to a set of resource constraints such as limited energy availability of SNs [12, 48], since in most cases there are only dependent to battery supply, limited network communication bandwidth and hardware-network heterogeneity. In a typical WSN, SNs are equipped with restricted computational power and limited amount of memory for data-signal processing and task scheduling. Furthermore, as SNs are usually deployed to hostile environments they are prone to failures and the replacement of failed SNs is practically impossible in case of large-scale WSNs or embedded sensors. Given the fact that WSNs can be used in security sensitive applications (e.g. battlefield surveillance, secure area monitoring and target detection [1]) the above-mentioned limitations represent critical challenges.

WSNs can be used in a variety of applications, including environment monitoring, automatic target detection and tracking, battlefield surveillance, remote sensing and global awareness [1]. A significant percentage of the above applications require remote retrieval of sensor readings and are known to be data intensive. An efficient method to reduce the volume of data communicated within a WSN is data fusion. In data fusion, readings from multiple SNs are combined and processed leading to more accurate data with significant smaller size, since redundant readings for neighbour SNs are filtered [61]. However suitable middleware solutions are required to utilize data fusion in WSNs. In this context, middleware is defined as a software layer, which functions as an interface between sensor nodes and the data fusion process [17]. Mobile agent (MA) technology has been proposed as an efficient middleware approach to IP networks, but is also suitable for WSN environments enabling the development and deployment of autonomic data fusion applications to resource constrained SNs. MAs are referred to as lightweight autonomic software entities that execute distributed tasks assigned by the users of a WSN, such as data fusion tasks.

The remainder of this chapter is organized as follows: First, we define the concept of data fusion, the related concepts of collaborative processing and data aggregation and the most representative approaches to WSN data fusion. We then present representative WSN middleware approaches and provide a comprehensive introduction to MA technology. Then we discuss the suitability, constraints and application fields of MAs in the context of WSNs. The next sections comprise the core of this chapter: we classify and review the most representative MA-based middleware proposals for autonomic data fusion in WSNs and evaluate their relevant strengths and shortcomings. Finally, we refer to open research issues in this field.

2 Data Fusion in WSN Environments

Data fusion is referred to as the process of combining data and knowledge from different sources with the aim of maximizing the useful information content [61]. It improves reliability while offering the opportunity to minimize the data retained.

Multi-sensor data fusion represents an evolving technology dealing with the problem of how to fuse data from multiple SNs to enable a more accurate estimation of the environment [45]. Such an approach achieves significant energy savings when intermediate SNs take part in the data fusion process (aggregate responses to queries). Applications of data fusion cross a wide spectrum, including environment monitoring, automatic target detection and tracking, battlefield surveillance, remote sensing and global awareness [1]. They are usually time-critical, cover a large geographical area and require reliable delivery of accurate information for their completion. Madden et al. in [37] discuss the implementation of five basic database aggregates, i.e. count, min, max, sum and average, based on the Tiny OS platform [57] and demonstrate that such a generic approach for data aggregation¹ leads to significant power (energy) savings. Other related works [21, 23, 33, 34] aim at reducing the energy expended by SNs during the process of data fusion. Most energy-efficient proposals are based on the traditional client/server computing model to handle multi-sensor data fusion in WSNs [19, 24, 28]; in that model, each SN sends its sensory data to a back-end PE or sink. However, as advances in sensor technology and computer networking allow the deployment of large amount of smaller and cheaper sensors, huge volumes of data need to be processed in real time.

To address the above problem a new concept, collaborative processing [67] has been introduced referring to cooperative data processing, where data are combined from multiple sources. This feature also differentiates a sensor network from traditional centralized sensing and signal processing systems where raw data are collected by SNs and then is routed, without prior processing, though the network to a central PE which carries out the whole processing. This client/server approach presents many problems such as large energy consumption due to the transmission of large amount of raw data. This is especially true for SNs around the central node which constantly should relay data from other SNs. Also, this approach consumes scarce bandwidth resources leading so to severe scalability problems specially when there is large number of SNs that should send data to the PE.

Collaborative processing takes advantage of the correlation inherent within the information of neighbor SNs. Since sensing regions are largely overlapping, data from neighbouring SNs refer to the same source of information (e.g. an evolving phenomenon, a moving target). Hence, an aggregation and/or fusion of the original sensory data is possible. This processing takes place as data pass through SNs and not at the edge of network in a powerful PE. This approach drastically reduces the communicated data and hence relieves the network from the huge amount of data that would otherwise should have been communicated and then collected to the PE. Some approaches utilizing collaborative processing according to Qi et al. [47] are as follows:

¹ Data aggregation is the process of refining data from multiple sensors into a summarization based on some rules or criteria. Examples of aggregation methods are statistical operations like the mean or the median.

1. The information-driven approach, which has initially developed for target tracking applications [7, 66]. This approach enables energy-efficient computing through selecting the next SN which most likely improves the tracking accuracy (based on information, cost and resource constraints). This application is built on directed diffusion² as the communication medium.
2. The relation-based approach [20] wherein the environment is sensed based on high-level description of the task and then instructs selected SNs to sense and communicate their sensory data.
3. The MA approach which provides more stable performance and improved fault tolerance than the information-driven approach, however, at the expense of extra bandwidth, needed for the transmission of the MA(s) code. This approach represents the main focus of this chapter.

3 Middleware Approaches in WSNs

The term middleware refers to the software layer positioned between the operating system and sensor applications on the one hand and distributed applications that interact with legacy systems on the other hand. The primary objective of the middleware layer is to hide the underlying complexity of the network environment by isolating the application from protocol handling, memory management, network functionality and parallelism [17].

The resource constraints (i.e. energy, limited memory and processing power) of contemporary nodes' hardware represent a challenge for the design of middleware solutions that meet the specific requirements of WSNs. Elen et al. in [8] categorized sensor middleware schemes into three main categories: application management where the middleware among other tasks has to deploy the application over the air on the WSN (i.e. Agilla [3, 13] and Mate' [31]), data management where the middleware has to handle the data packets that flows though the network (i.e. Milan [22], TinyDb [38] and Dsware [32]) and network service management where the middleware should offer a set of network services to the 'running' applications (i.e. Impala [35]).

MAs represent a promising middleware approach and in fact have been already used in some middleware schemes [3, 13]. A WSN mobile-agent middleware system (WMMS) must provide a platform to support MAs to perform user-assigned tasks (in our case autonomic data fusion tasks) while enabling the application deployment in WSNs.

² Directed diffusion [2, 7, 54] is a novel network protocol built for information retrieval and data dissemination. Its main characteristic is that it is 'data-centric', namely routing is based on data aggregated in the SNs rather than traditional IP theme where end-to-end delivery method is used based on unique identifications. Data generated by SNs are named by attribute-value pairs. A node requests data by sending interests for named data. Data matching the interest is then 'drawn' down towards that SN (set up of gradients). Intermediate SNs can cache or transform data and may direct interests based on previously cached data.

4 Brief Introduction to Mobile Agent Technology

MA technology represents a relatively recent trend in distributed computing, which answers the flexibility and scalability problems of centralized models. The term MA [42] refers to an autonomous program with the ability to move from host to host and act on behalf of users towards the completion of an assigned task. In addition, they are able to interact with legacy systems. Recently, MAs have been proposed for efficient data dissemination in sensor networks [4, 45, 46, 56, 64, 65]. In a traditional client/server-based computing scheme (see Fig. 1), data from multiple nodes is transferred to one destination. Because the bandwidth of a WSN is typically much lower from other types of networks (e.g. wired networks), the data traffic derived from remote interactions may soon exceed the network capacity of a WSN. This serious scalability problem may be sufficiently addressed through MAs. The key idea is to delegate multiple MAs to the SNs of a WSN (see Fig. 1); these MAs can perform local rather than remote interactions with legacy systems, apply intelligent filtering operations thereby eliminating redundancy in the transferred data and decreasing the network overhead associated with data transfer. This approach also implies more reasonable usage of the nodes' radio unit (i.e. their most energy consuming part), hence prolonged nodes and network lifetime.

A MA may be defined as an entity of four attributes (see Fig. 2) [46]:

- Identification: a number used to uniquely identify the MA in the format of two-tuple $(i : j)$, where i indicates the IP address of the dispatcher and j the serial number assigned to agents by the dispatcher.
- Data space: the agent's data buffer which carries the partially integrated results (this result should provide progressive accuracy as the agent migrates from node to node).
- Itinerary: the route of migration. Itinerary planning includes two main issues that must be addressed: (a) the selection of SNs that must be visited and (b) the route

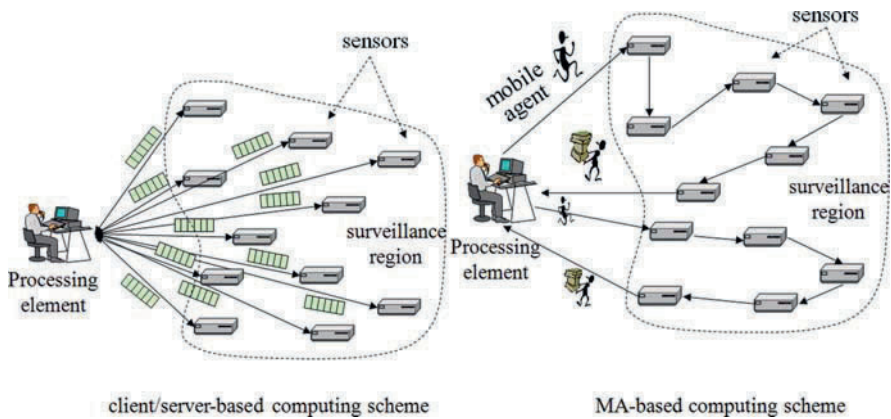


Fig. 1 Client/Server and MA-based computing schemes

that the MA will follow to visit the selected SNs. Itinerary scheduling can be classified as dynamic, static or hybrid.³

- **Methods:** the application logic (or execution code) carried with the agent.

5 Advantages/Disadvantages and Application Fields of MAs in the Context of WSNs

Lange and Oshima listed seven good reasons to use MAs [30]: reducing network load, overcoming network latency, robust and fault-tolerant performance, etc. The MA-based computing model enables moving the code (processing) to the data rather than transferring raw data to the processing module. By transmitting the computation engine instead of data, this model offers several important benefits:

- **Decreased network overhead.** Instead of passing large amounts of raw data over the network through several round trips, only an MA of small size is deployed. This is especially important for real-time applications and whenever the communication is enabled through low-bandwidth wireless connections. This also substantially implies improved network scalability.
- **Distribution of network load.** In the client/server model, the network area around the centralized element represents a bottleneck point (all remote calls are initiated and returned to this element). MAs favour network load balancing since agents are dispatched from the PE in the beginning of their journey and return back in the end of their itinerary. In the meanwhile, agent migrations typically occur in network regions away from the PE.
- **Adaptability.** MAs can be programmed to carry task-adaptive processes which may extend the built-in capability of the system on the fly.
- **Stability and fault tolerance.** MAs can be dispatched when the network connection is alive and return results when the connection is re-established. Therefore, the performance of the system is not much affected by the reliability of network links.
- **Autonomy.** An agent is a self-contained software element responsible for performing part of a programmatic process. Therefore, it contains some level of intelligence, ranging from simple predefined rules to self-learning artificial intelligence (AI) inference machines. It acts typically on behalf of a user or a process enabling task automation. MAs operate rather autonomously (they are often event or time triggered) and may communicate with the user, system resources and other MAs as required to perform their task. The autonomy feature of MAs

³ A dynamic itinerary is determined on the fly at each hop of the MA, while a static itinerary is computed at the PE prior to the MA migration. In a hybrid approach, the SNs to be visited are selected by the PE but the visiting order is decided on the fly by the MA. Although improving the optimality of the MA's itinerary compared to hybrid and static approaches, the dynamic approach is more time expensive (the 'find-next-node' function is executed on each migration step), consumes valuable sensor nodes energy resources and implies larger MA sizes (the more intelligence integrated within the agent, the larger its size).

mainly refers to their ability to exercise control over their own actions. That is, to respond in a timely fashion to changes in the environment, to exhibit goal-oriented behaviour by taking the initiative and possibly change their itinerary on-the-fly depending on how they perceive their environment. In fact, autonomy has been agreed to comprise – among others – an essential feature of mobile agents [26, 29]. This feature along with platform and system independence makes them ideal for building robust and reliable WSNs. Furthermore MAs can support WSNs deployed in hostile environments, since they react dynamically to changes (for example interference or jamming).

Although the role of MAs in distributed computing is still being debated mainly because security concerns [15], several applications have shown clear evidence of benefiting from the use of MAs [40], including e-commerce and m-commerce trading [53], distributed information retrieval [25], network awareness [18] and network and systems management [15, 49, 50]. Network-robust applications are also of great interest in military situations today. MAs are used to monitor and react instantly to the continuously changing network conditions and guarantee successful performance of the application tasks.

MAs have also found a natural fit in the field of distributed sensor networks (DSNs); hence, a significant amount of research has been dedicated in proposing ways for the efficient usage of MAs in the context of WSNs. In particular, MAs have been proposed for enabling dynamically reconfigurable WSNs through easy development of adaptive and application-specific software for SNs [58], for separating SNs in clusters [36], in multi-resolution data integration [56] and fusion [46], data dissemination [5] and location tracking of moving objects [2, 56, 63]. These applications involve the usage of multi-hop MAs visiting large numbers of SNs.

In MA-based autonomic data fusion tasks the choice of agents’ itineraries is of critical importance affecting the overall energy consumption and data fusion cost.

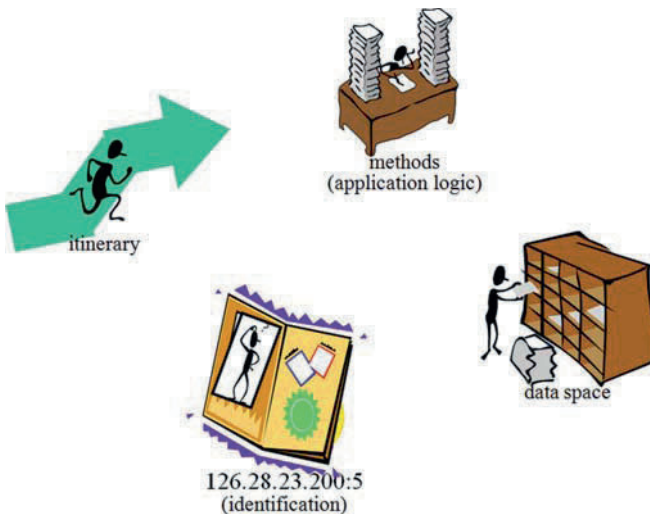


Fig. 2 Mobile agents as entities of four attributes

Notably, only few research articles have dealt with the problem of approximating optimal MA routes either through heuristics [45] or genetic algorithms [64]. The most notable weakness of these algorithms as argued in Sect. 7.1 is that they rely on a single MA to visit and fuse data from distributed sensors. However, such solutions do not scale acceptably for networks comprising hundreds or thousands of sensor nodes.

Security has been identified as the main reason that hinders the adoption of MAs as the next generation distributed computing paradigm.⁴ In the context of WSNs, the most crucial security risk in using MAs is the possibility of tampering an agent. In a WSN that utilizes MAs the agent's code and internal data autonomously migrate between SNs and could be easily changed during the transmission or at a malicious (hostile) node. To address this security risk several countermeasures can be utilized to detect any manipulation made by an adversary, for instance, Encrypted Functions (EF) [51], Cryptographic Traces [59, 60], Chained MAC protocol [11], Watermarking [10], Fingerprinting [10], Zero-knowledge proofs [43] and the Secure secret sharing scheme [43].

6 Classification of MA-Based Approaches for Autonomic Data Fusion in WSNs

MA-based approaches for autonomic data fusion in WSNs can be classified as follows (see Fig. 3):

- Single MA-based autonomic data fusion [5, 6, 45, 64], where only one MA is used for autonomic data fusion in the sensor network. This approach may work well with small scale WSNs but as discussed in Sect. 5 such solutions do not scale acceptably for networks comprising hundreds or thousands of sensor nodes.
- Multiple MA-based autonomic data fusion [16, 41, 46, 56], wherein a number of MAs is working in parallel to fuse data from WSN sensors. This approach is highly efficient even in large scale WSNs. However, it requires relatively complex algorithms to derive the itineraries of individual MAs.
- Autonomic data fusion in clustered WSN architectures [61]. This category refers to sensor networks with clustered structures. In such structures, SNs located in nearby locations are grouped in virtual clusters; one of these SNs acts as 'cluster head' (CH) and sensory data retrieved by its 'cluster members' are routed to the PE through the CH. Autonomic data fusion tasks within these clustered structures are assigned to MAs.

⁴ The most critical security concerns related to MAs comprise (a) protecting mobile hosts from malicious agents, (b) protecting agents from malicious hosts and (c) protecting sensitive information carried by agents from eavesdropping. The first is addressed through implementing authentication and authorization features which ensure that only trusted agents may be executed, in a restricted authority domain. The second may be achieved through protecting agents against tampering, while the third is sufficiently addressed through encrypting sensitive data.

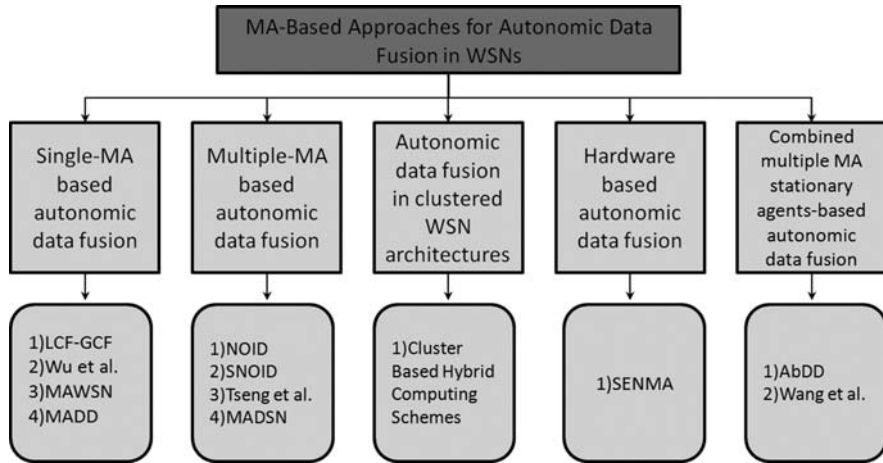


Fig. 3 A taxonomy of MA-based approaches for autonomic data fusion in WSNs

- Hardware-based autonomic data fusion [55]. In Sect. 4 we described MAs as autonomous software entities. Several research papers though used the term MA to refer to mobile hardware instances programmed with suitable software and acting as MAs (e.g. highly mobile SNs traversing the WSN to collect and process data from SNs and deliver it back to the PE).
- Combined multiple MA / stationary agents-based autonomic data fusion [39, 63]. This approach involves SNs with embedded stationary agents (SAs) that cooperate with their peers of neighbour nodes or MAs that traverse the network.

7 MA-Based Approaches for Autonomic Data Fusion in WSNs

In this section we present the key concepts of the most representative MA-based autonomic data fusion schemes found in the literature, emphasizing on their relevant merits and shortcomings. The presentation is structured according to the classification provided in the preceding section (Sect. 6).

7.1 Single MA-Based Autonomic Data Fusion

7.1.1 Local Closest First and Global Closest First algorithms

Qi and Wang in [45] proposed two heuristics to optimize the itinerary of MAs involved in data fusion tasks. In local closest first (LCF) algorithm, each MA starts its route from the PE and searches for the next destination with the shortest distance to its current location. In global closest first (GCF) algorithm, MAs also start their itinerary from the PE node and select the node closest to the centre of the surveillance region as the next-hop destination.

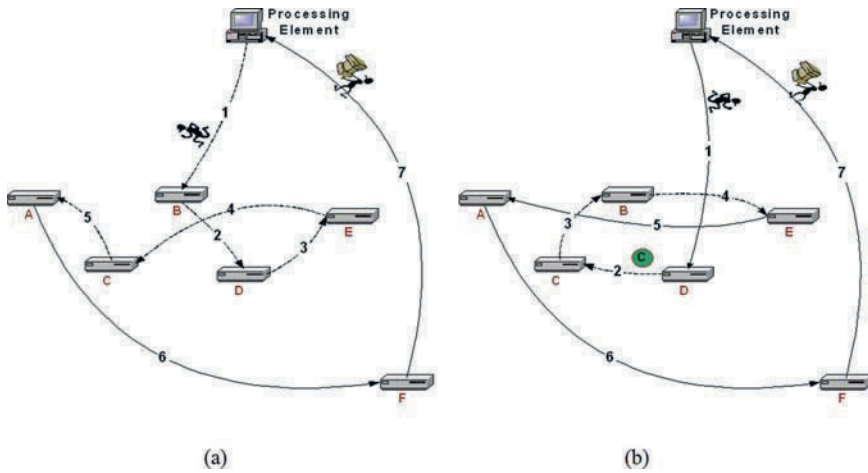


Fig. 4 (a) Output of LCF, (b) output of GCF ('C' denotes the network's centre)

The main asset of these two algorithms is that they are associated with low computational complexity. However, the output of LCF-like algorithms highly depends on the MAs original location, while the nodes left to be visited last are typically associated with high migration cost [27] (see, for instance, the last two hops, 6 and 7, in Fig. 4a); the reason for this is that they search for the next destination among the SNs adjacent to the MA's current location, instead of looking at the 'global' network distance matrix. On the other hand, GCF produces in most cases messier routes than LCF and repetitive MA oscillations around the region centre, resulting in long route paths and unacceptably poor performance [45, 64].

The most serious drawback in both LCF and GCF is that they involve the use of a single MA object launched from the PE station that sequentially visits all SNs, regardless of their physical location on the plane. Their performance is satisfactory for small WSNs; however, it deteriorates as the network size grows and the sensor distributions become more complicated. This is because both the MA's round-trip delay and the overall migration cost increases squarely with network size, as the travelling MA accumulates into its state data from visited SNs [14]. The growing MA's state size not only results in increased consumption of the limited wireless bandwidth, but also consumes the limited energy supplies of SNs. This drawback is addressed by more recent works [16, 40] that propose methods for intelligent itinerary scheduling enabling the parallel employment of multiple MAs, each visiting a limited number of nodes.

7.1.2 Wu et al. Genetic Algorithm

Wu et al. proposed a genetic algorithm⁵ for computing routes for a MA that incrementally fuses the data as it visits the nodes in a WSN [64]. The main application

⁵ A genetic algorithm is a computational mechanism that 'simulates' the process of genetic selection and natural elimination in biological evolution.

fields of this approach are object tracking and detection, where the MA must visit the sensors that sense the strongest signals, while the suggested itinerary must keep path loss and energy consumption low. The authors proved the above route computational problem to be NP-hard. Hence they relied on a genetic algorithm to solve the problem.

A two-level encoding is employed to adapt the genetic algorithm for the MA routing problem in WSNs. The first level is a numerical encoding of the sensor (ID) label in the order of SNs being visited by the MA. The second level is a binary encoding of the visit status of the SNs that are used in the first level (e.g. if a sensor has been visited, it is assigned '1' attribute value, else '0').

The proposed genetic algorithm is based on an event-driven adaptive method to implement a semi-dynamic routing strategy where the routing code is implemented exclusively in the PE. The MA carries only the pre-computed route that determines the order of SNs to be visited. In case of topology changes in the network (e.g. loss of nodes communication or energy depletion) that render the previous computed route invalid, the routing code is re-executed at the PE and the new route is sent to the MA.

Although providing superior performance (lower itinerary cost) than LCF and GCF algorithms, this approach implies a time-expensive optimal itinerary calculation (genetic algorithms typically start their execution with a random solution 'vector' which is improved as the execution progresses), which is unacceptable for time-critical applications, e.g. in target location and tracking. Also, in such applications, the group of visited SNs (i.e. those with maximum detected signal level) is frequently changed over time depending on target's movement; hence, a method that guarantees fast adaptation of MAs itineraries is needed.

7.1.3 Mobile Agent-Based Wireless Sensor Network Architecture

Chen et al. in [5] proposed the mobile agent-based wireless sensor network (MAWSN) architecture for filtering and aggregating data in planar sensor network architectures. In MAWSN, MAs are used to (a) eliminate data redundancy among SNs through applying context-aware local processing at the node level; (b) eliminate spatial redundancy among neighbour SNs by MA assisted data aggregation, since in WSNs comprising large numbers of SNs closely located sensors generating redundant data are likely to exist and (c) reduce communication overhead by using a packet unification technique that concatenates the data from several short packets into one longer packet in order to reduce the communication overhead at the combined task level.

In MAWSN it is assumed that the PE is aware of the nodes that will be visited by the MA and the itinerary of the MA is predetermined. The payload of an MA is consisted of two parts, the processing code which is used to process sensed data and the aggregated data. Also the MA keeps a list (*SourceList*) with the source nodes that has to visit. In *SourceList*, there are two sources whose positions are important, namely, the first source which the MA will visit (*FirstSource*) and the last source (*LastSource*). The pair of *FirstSource* and *LastSource* represents the starting and

ending points of the MA, respectively, while *Nextsource* represents intermediate nodes.

When an MA is dispatched from the PE, it visits *FirstSource* where it is stored. *FirstSource* dispatches a copy of the stored MA (clone) after specific periods which are predefined according to the desired data rate. The clone after leaving *FirstSource* visits *Nextsource* nodes according to their gradient [54] (each time selects the one with maximum gradient), collects sensed data and deletes the current *NextSource* node from its *SourceList*. After visiting all *NextSource* nodes the clone finally reaches *LastSource*, where it also collects sensed data and then returns to the PE.

The authors proved via simulations that MAWSN presents performance gain over the client/server model in terms of energy consumption and packet deliver ratio. However, as the authors admit, MAWSN involves longer end-to-end latency under certain conditions due to the fact that only a single MA is employed in MAWSN. In scenarios wherein the MA visits large sets of sensors the latency (round trip delay of the MA) and the energy expenditure is drastically increased.

7.1.4 Mobile-Agent-Directed Diffusion Architecture

Chen et al. proposed the mobile-agent-directed diffusion (MADD) in [6] as an improvement to MA-based distributed sensor network (MADSN) and multi-resolution integration (MRI) algorithm (see Sect. 7.2.4). The limitation of clustering in MASDN is addressed by using a flat network architecture (the authors argue that is more suitable for a wide range of WSNs applications compared to cluster-based architectures).

In MADD, MAs itineraries are scheduled by the PE utilizing directed diffusion [54]. The itinerary scheduling is the same with MAWSN with the only difference that when the clone MA reaches *LastSource* it discards the processing code and carries only the aggregated result to the PE saving valuable energy. The main differences between MADD and client/server model are as follows:

- MADD uses a single MA that visits all the relevant SNs to collect data and the interval between the reports to the PE is decided by the dispatching rate of the MA. On the contrary, in client/server-based WSNs the sensory data is sent individually by each sensor with a specified interval.
- In MADD, data is collected by the MA visiting all the SNs along a single path (itinerary), while in client/server-based WSNs, data is sent back to the PE in parallel from all nodes.

Although MADD addressed many constraints of MASDN, it failed to address the poor scalability of the approach wherein a single MA object visits sequentially the SNs for data collection. This renders both algorithms inappropriate for large-scale WSNs wherein the end-to-end delay and the size of the MA would increase squarely with the number of visited SNs.

7.2 Multi-MA-Based Autonomic Data Fusion

7.2.1 The Near-Optimal Itinerary Design Algorithm

Mpitzopoulos et al. in [41] proposed the near-optimal itinerary design (NOID) algorithm to address the problem of calculating a near-optimal route for a MA that incrementally fuses the data as it visits the nodes in a DSN.

NOID algorithm adapts some basic ideas of Esau–Williams (E–W) algorithm [9] and has been designed on the basis of three objectives: (a) MA itineraries should be derived as fast as possible and adapt quickly to changing networking conditions (hence, an efficient heuristic is needed), (b) MA itineraries should include only SNs with sufficient energy availability and exclude those with low energy level and (c) the number of MAs involved in the data fusion process should depend on the number and the physical location of the SNs to be visited; the order an MA visits its assigned nodes should be computed in such a way as to minimize the overall migration cost.

As opposed to single MA-based approaches, NOID enables the construction of multiple near-optimal itineraries, each assigned to individual MAs (see Fig. 5). NOID is executed on the PE platform; hence, MA routes are predefined and not computed on-the-fly (awareness of the nodes’ geographical locations is assumed). The authors claim this is a reasonable choice since MAs starts their journey from the PE node, which is typically equipped with powerful computing resources compared to SNs. However, a dynamic itinerary calculated at each hop by the MA would enable more prompt response to potential topology changes. On the other hand, it would rise energy demands since the next-hop computation would execute on resource-constrained SNs; in addition, the MAs size would considerably increase

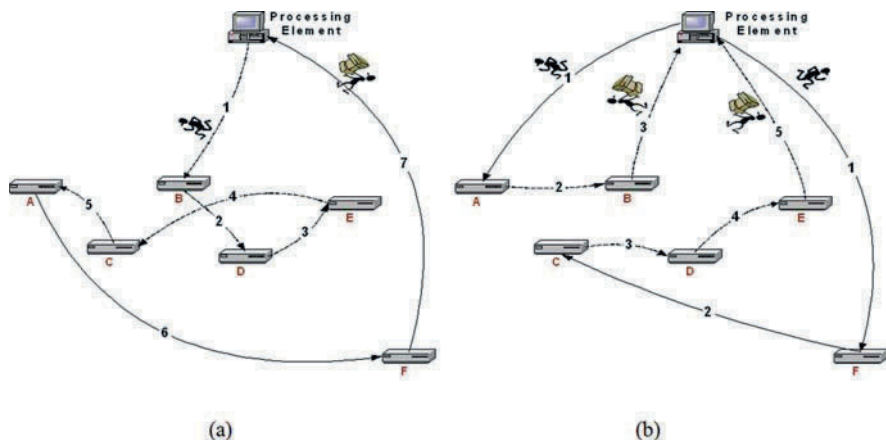


Fig. 5 (a) Output of NOID (the sequence numbers indicate the order in which the corresponding MA migrations are accepted, i.e., the algorithm’s iteration sequence numbers), (b) the MA itineraries derived from the NOID algorithm’s output

(the itinerary scheduling logic would be embedded into MA code and transferred on every MA migration).

The authors also reported simulation results that demonstrated the improved NOID's performance over LCF and GCF algorithms in terms of the overall energy consumption and response time. This is mainly because NOID takes into account the amount of data accumulated by MAs at each visited SN. Namely, NOID recognizes that travelling MAs become 'heavier' while visiting SNs without returning back to the PE to 'unload' their collected data [14]. Therefore, NOID promotes small itineraries enabling the parallel employment of multiple cooperating MAs, each visiting a subset of SNs.

7.2.2 The Second Near-Optimal Itinerary Design Algorithm

Gavalas et al. in [16] presented the second near-optimal itinerary design (SNOID) algorithm for determining the number of MAs that should be used and the itineraries these MAs should follow.

The main idea behind SNOID is to partition the area around the PE into concentric zones and start building the MA paths with direction from the inner (close to PE) zones to outer ones. The radius of the first zone which includes the PE is equal to $a \times r_{\max}$ where a is an input parameter in the range $(0, 1]$ and r_{\max} is the maximum transmission range of any SN. All SNs inside the first zone are connected directly to the PE and these nodes are the starting points of the itineraries of the MAs. By adjusting the value of parameter a , the number of MA itineraries is adjusted accordingly.

Summarizing, SNOID algorithm determines the number of MAs by taking only into account the cost of communication between the PE and the first nodes of the itineraries. The remaining zones have a constant width equal to $r_{\max}/2$. Thus, each SN can only directly communicate with nodes residing within the same zone or within the two adjacent zones.

The construction of the itineraries of the MAs starts from the inner (close to PE) zones and proceeds to the outer zones, as illustrated in Fig. 6. When examining a zone, SNOID's objective is to connect each node with a node of the previous or the current zone which already has a connecting path back to the PE. Throughout this process attention is paid to the latency (denoted as PL in Fig. 6) of the trees formed up to that point, where latency is calculated through a simple formula. The nodes selected to join the trees are the ones that provide the minimum cost to the tree compared to other candidate nodes.

7.2.3 Location Tracking in a WSN by MAs and its Data Fusion Strategies

In [56] Tseng et al. proposed the use of MAs for location tracking applications in WSNs to reduce sensing, computing and communication overheads. When a new object is detected by a SN, an MA is initiated to track the roaming path of the object. The MA visits the SN closest to the object and hops to other SNs following the object's movement. This MA (called master MA) may invite two nearby SNs

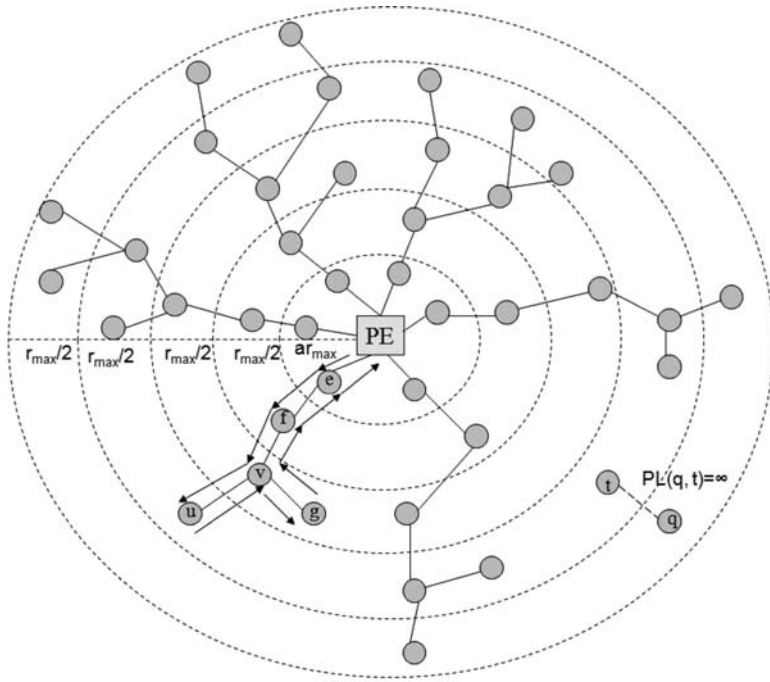


Fig. 6 Partitioning the area around PE into a number of zones (SNOID)

to cooperatively position the object by dispatching a slave MA to each of them. Following that, the three MAs (the master MA and the two slave MAs) cooperate to perform the trilateration⁶ algorithm [52] to calculate the object’s precise location. As the object moves, the slave MAs may be revoked or reassigned depending on how ‘strongly’ they sense the moving object. Regarding the number of slave MAs the authors point out that although their development is based in the cooperation of only two, it will be straightforward to extend their work to more slave MAs to improve the positioning accuracy.

Besides location tracking the authors try to address the problem of fusion of data containing the tracking results. They propose two schemes to transfer the fused results to the PE, the threshold-based (TB) scheme and the distance-based (DB) scheme. In TB scheme the results are forwarded to the PE when the data size carried by the MA(s) reaches an upper bound while in DB scheme both data size and the distance of the MA from the PE are considered. Simulation results proved that DB performs well in all cases while in TB the threshold should be carefully chosen.

⁶ Trilateration is a method of determining the relative positions of objects using the geometry of triangles in a similar fashion as triangulation. It uses the known locations of two or more reference points, and the measured distance between the subject and each reference point.

7.2.4 MA-Based Distributed Sensor Network and MRI Algorithm

To solve the problem of the overwhelming data traffic, Qi et al. [46] proposed the MADSN by developing an enhanced version of the original MRI algorithm [44] for scalable and energy-efficient data aggregation. The idea of the original MRI algorithm [44] involves the construction of a simple function (overlap function) from the outputs of the sensors in a cluster and resolving this function at various successively finer scales of resolution to isolate the region over which the correct sensors lie. Each sensor in a cluster measures the same parameters. It is possible that some of them are faulty. Hence it is desirable to make use of this redundancy of the readings in the cluster to obtain a correct estimate of the monitored parameters.

The key concept in the enhanced version of MRI algorithm [46] for MADSNs is that multi-resolution analysis is applied at each sensor node instead of the PE, allowing MADSN to save up to 90% of data transfer time, according to the simulations the authors have conducted, compared to the original MRI [44] implementation for DSNs.

Concluding, in this approach by transmitting the software code (MA) to SNs that are nearby the area(s) of interest, a large amount of sensory data may be filtered at the source by eliminating the redundancy utilizing the enhanced MRI algorithm [46]. MAs may visit a number of SNs and progressively fuse retrieved sensory data, prior to returning back to the PE to deliver the data. This scheme may be more efficient than traditional client/server model; within the latter model, raw sensory data are transmitted to the PE where data fusion takes place.

The main drawback of MADSN is that it operates based on the following assumptions [6]: (1) the sensor network architecture is clustered; (2) source nodes are within one hop from a CH and (3) much redundancy exists among the sensory data which can be fused into a single data packet with a fixed size. These assumptions pose many limitations on the range of applications that may be supported by MADSN. Also in [46] the authors assume a constant size for the MA, which is a non-valid assumption since MAs may get ‘heavier’ from the data they collect as they migrate through SNs.

7.3 *Autonomic Data Fusion in Clustered WSN Architectures*

7.3.1 Cluster-Based Hybrid Computing Schemes

Clustered architectures involve specific SNs operating as CHs; such nodes are typically positioned in their cluster’s centre, while they act as relays for forwarding the sensory data retrieved by their assigned cluster members to the Sink. Xu and Qi in [61] argued that MA-based WSNs does not necessarily perform better than client/server schemes since MAs also introduce overhead due to their migrations and access to legacy systems resources.

Along this line, they proposed two cluster-based hybrid computing schemes that combine the advantages of MA and client/server models and offer better performance, should the proper scheme is chosen according to network clustering condi-

tions. Within the first scheme (scheme A) each CH dispatches an MA that visits all the cluster members (SNs) in sequence to collect and aggregate data. When an MA returns to the CH, it sends the aggregated data back to the PE. In the second scheme (scheme B) an MA is dispatched by the PE and visits the CHs to collect the sensory data retrieved by the associated cluster members through client/server interactions.

The main drawback of the proposed hybrid model, as admitted by the authors, is that while for some network configurations it can make full use of the advantages of both the client/server and MA models, for some other configurations it could inherit the disadvantages of both models, leading to decreased performance. An example of such a situation is when the number of clusters is large for scheme A and small for scheme B. Also the authors in [61] assume a constant size for the MA, which is not realistic as the MA grows ‘bigger’ while collecting data from the SNs.

7.4 Hardware MA-Based Autonomic Data Fusion

7.4.1 SENMA

Tong et al. in [55] proposed an architecture for large scale low-power sensor network, called sensor networks with mobile agents (SENMA) architecture. SENMA exploits node redundancies by introducing MAs that visit the SNs periodically or when the application requires for data gathering or network maintenance. The addition of MAs shifts computationally intensive tasks away from primitive SNs to more powerful MAs enabling energy efficient operations under severely limited power constraints.

MAs in SENMA are powerful hardware units equipped with sophisticated transceivers. Compared to regular SNs these special MAs are not strictly constrained on their communication and processing capability and their ability to traverse the sensor network. Examples of MAs could be manned/unmanned aerial vehicles, ground vehicles equipped with sophisticated terminals and power generators, or specially designed light nodes that can hop around in the network [55].

The authors showed that the simple topology of SENMA reduces energy consumption and improves the scalability of WSNs [55]. The main drawback of SENMA is the requirement of special hardware, playing the role of MAs that would significantly increase the cost and the deployment complexity of the WSN. However, SENMA could be efficiently used in special applications where the cost of the WSN is not a first-priority issue (e.g. military applications).

7.5 Combined MA/Stationary Agents-Based Autonomic Data Fusion

7.5.1 Agent-Based Directed Diffusion

In [39] agent-based directed diffusion (AbDD) is proposed by Malik et al. to address one of the most significant drawbacks in current routing schemes for WSNs: they all

tend to propose optimal route that consume lowest energy (e.g. minimum number of hops path), leading all the nodes along the optimal path to faster energy depletion. On the contrary, AbDD ensures that data routing traffic is fairly balanced across the network as it takes into account both the routing cost and remaining energy of the nodes and utilizes the cloning capability of MAs. AbDD uses directed diffusion to address the redundancy of sensory data especially in large-scale dense WSNs with arbitrary nodes placement. Apart from MAs, SAs permanently residing to SNs are also used (SAs directly interact with legacy systems, retrieve sensory data and report them to the MAs).

MAs and SAs in conjunction with directed diffusion achieve energy saving by moving the processing function to the data rather than transferring the data to the PE. AbDD's execution encompasses the following phases:

- Phase 1: The PE dispatches a single MA equipped with specified interest for data to identify – cooperating with the local SAs – the SNs that satisfy these interests (termed source SNs). In effect, an interest message is a query or an interrogation which specifies what the PE looks for. Each interest contains a description of sensing task that is supported by the WSN for acquiring data. When the MA locates a SN that meets the specified interests, it returns back to the PE building on the way the cost tables.
- Phase 2: The PE dispatches the MA to distribute the application-specific code to the source SNs. The sequence of source SNs to be visited is predetermined by the PE. However, MAs may alter their itinerary on-the-fly taking into account the battery level for each neighbour SN at each hop. When the MA reaches the first source SN, it distributes the copy of application code to the SA and clones itself to continue its itinerary. Next, it continues its itinerary and distributes copies of the application code at each source SN. When it reaches the last source SN is self-destroyed.
- Phase 3: The MA's clone residing in the first source SN makes a new clone that remains to the node (when a time interval set by the PE elapses). It then collects the data from the sensor and migrates to the other source SN performing data aggregation functions. Finally, it routes data back to the PE and destroys itself.

The authors proved through simulations that agent-based directed diffusion achieves lower energy consumption than distributed directed diffusion. However, although AbDD utilizes the cloning capability of MAs only a single MA initially is dispatched by the PE and a single MA visits the source SN and reports back to the PE. Thus, similarly to alternative single MA-based data fusion schemes AbDD is associated with increased energy consumption and latency when considering data retrieval from large sets of SNs.

7.5.2 Agent Collaborative Target Localization and Classification in Wireless Sensor Networks

Wang et al. in [63] proposed a heterogeneous architecture for WSNs, for target localization and classification tasks where both multi-agent systems (each node

represents a multi-agent) and MAs are incorporated. The multi-agent system comprises a four-level hierarchy [63], where the top level is the interface agent. This agent is responsible for three procedures, namely receiving user queries about the environment, forwarding these queries to the lower level agents accordingly and reporting the query results to the users. In the immediately lower level resides the regional agent. According to the size of the WSN more than one regional agents may coexist, with each one being in charge of a region within the WSN. The task of the regional agents is to receive query requests from the interface agent and to coordinate sensor nodes within their region to collaboratively respond to the requests. A region is further split to clusters that are managed by manager agents. The role of manager agents is to directly control the behaviour of sensor nodes (observing agents) that are given the task of sensing the current target event.

In the above hierarchical multi-agent system, MAs are employed only when necessary and beneficial (e.g. in cases where data transmission comes in bulk or utilization of MAs gives superior performance). Sensor fusion is primarily performed based on multi-agent cooperation, however MAs are also used in cases of bulk data exchanges. This architecture greatly facilitates designs and implementations of WSN. In addition the architecture also readily adapts to diversified deployments at various scales.

Summarizing, the main purpose of this work is to develop the appropriate data fusion mechanisms, which should provide desirable accuracy and at the same time adapt to various WSN constraints (e.g. limited bandwidth and energy). They take advantage of both multi-agent and multi-MA schemes to achieve their goals.

8 Comparison of MA Approaches for Autonomic Data Fusion in WSNs

The performance of algorithms that fall into single MA-based category is relatively low especially in large-scale WSNs. This is because a single MA is used for data fusion tasks and, hence, it must carry a heavy load of data retrieved from SNs. This category is only suitable for small-scale WSNs. On the other hand, multi-MA based category can be efficiently used in large-scale WSNs, since multiple MAs are working in parallel for data fusion tasks leading to considerable energy gains. However, the complexity of the proposed algorithms is increased compared to single MA-based algorithms since they cater for scheduling non-overlapping itineraries for multiple MAs.

In clustered architectures the use of MAs for data fusion tasks may result in fast energy depletion of CH nodes. To overcome this limitation, clustering algorithms typically cater for the election of new CHs in periodic basis. However, that increases the complexity of MAs routing since the itinerary scheduling algorithm should be aware of the current CHs identity on every execution. Furthermore,

this category has limited applicability to WSNs consisted of mobile SNs since their topology is frequently modified. Finally, CHs represent bottleneck points within the WSN, especially when transfer of large chunks of sensory data is involved.

The proposals in hardware-based category can offer comparable or improved performance than multi-MA-based data fusion methods. Thus, they can efficiently be used in large-scale WSNs where cost is not a prohibitive factor. Proposals that belong to combined MA/stationary agents-based category can also be used in large-scale WSNs because multiple MAs and SAs are utilized in data fusion process. These approaches offer comparable performance to that of multi-agent-based models, however they imply complex manageability since they involve MA and SA entities. In addition, the permanent execution of SAs upon SNs implies increased demand upon device resources.

In Fig. 7 we summarize the advantages and disadvantages of all the research works reviewed in this chapter while Fig. 8 lists various parameters such as application field, number of utilized MAs and relevant category. We also refer to the parameters that affect the overall performance of each work (sensory data redundancy ratio and MA initial size applies to all approaches) and where the itinerary planning of MA(s) takes place.

	Pros	Cons
LCF - GCF	<ul style="list-style-type: none"> • Low complexity • Easy to implement 	
MAWSN	<ul style="list-style-type: none"> • Utilization of spatial redundancy • Reduction of communication overhead 	<ul style="list-style-type: none"> • Poor scalability: Increased energy consumption and latency in large scale WSNs
MADD	<ul style="list-style-type: none"> • Suitable for flat network architectures 	
Wu et al.	<ul style="list-style-type: none"> • Superior performance compared to LCF-GCF 	<ul style="list-style-type: none"> • Time-expensive itinerary computation (not adequate for time critical fusion tasks)
NOID	<ul style="list-style-type: none"> • Suitable for large scale WSNs (reduced communication overhead, decreased response time) • Takes into account nodes energy availability 	<ul style="list-style-type: none"> • Increased complexity • The PE must know the position of each SN in the field
SNOID	<ul style="list-style-type: none"> • Suitable for large scale WSNs (reduced communication overhead, decreased response time) • Takes into account nodes communication range 	
Tseng et al.	<ul style="list-style-type: none"> • Takes into account load of data carried by the MA • Takes into account spatial distance MA-PE 	<ul style="list-style-type: none"> • Increased complexity
MADSN	<ul style="list-style-type: none"> • Suitable for large scale WSNs with clustered architecture 	<ul style="list-style-type: none"> • Suitable only for clustered WSN architectures • Source nodes are assumed one hop away from a CH (one-hop clusters) • Assumes increased redundancy of sensory data
Cluster based hybrid computing schemes	<ul style="list-style-type: none"> • Uses the best computing model (among MA-based and client/server) depending on the network configuration • Inherits advantages of client/server and MA models 	<ul style="list-style-type: none"> • Suitable only for clustered WSN architectures • In some WSN configurations it inherits the disadvantages of both client/server and MA models.
SENMA architecture	<ul style="list-style-type: none"> • Suitable for special WSN scenarios • Increased security of MAs 	<ul style="list-style-type: none"> • Increased implementation cost • Need for specialized hardware
Abdd scheme	<ul style="list-style-type: none"> • Increased WSN lifetime 	<ul style="list-style-type: none"> • Complex manageability (both MAs and SAs involved)
Wang et al. scheme	<ul style="list-style-type: none"> • MAs are only used when needed reducing overhead • Scalable architecture 	<ul style="list-style-type: none"> • Increased demand upon device resources (SAs permanently reside on SNs)

Fig. 7 Advantages–disadvantages of each proposed scheme

	Application(s) field	Category	Parameters that affect the overall performance	Itinerary planning
LCF	data fusion	single-MA based	network scale	static
GFG	data fusion	single-MA based	network scale	static
Wu et al.	data fusion	single-MA based	network scale	hybrid
MAWSN	data fusion-aggregation	single-MA based	network scale	hybrid
MADD	data diffusion	single-MA based	network scale	static
NOID	data fusion	multiple-MA based	number of dispatched MAs	static
SNOID	data fusion	multiple-MA based	number of nodes within the PE range	static
Tseng et al.	location tracking	multiple-MA based	data size carried by the MA, spatial distance MA-PE	dynamic
	data fusion			
MADSN	data fusion-aggregation	multiple-MA based	number of dispatched MAs	static
Cluster based hybrid computing schemes	data fusion	clustered WSN	number of clusters, average cluster size	static
SENMA architecture	data fusion-aggregation	hardware based	number of dispatched MAs	dynamic
Abdd	data fusion-aggregation	combined multiple MA/SA based	network scale	hybrid
	data diffusion			
Wang et al.	location tracking	combined multiple MA/SA based	number of dispatched MAs	dynamic
	data fusion		number of clusters	

Fig. 8 Individual parameters of of each proposed scheme

9 Open Research Issues

The rapid advances in sensor technology depress the manufacturing cost of SNs and make feasible the deployment of large-scale WSNs. However the increased number of SNs implies increased data volumes that must be transferred through the limited bandwidth of wireless channels. This problem along with the resource constraints of contemporary SNs (e.g. limited energy, computation and communication capabilities) raises new challenges to the design of scalable and functional WSNs. To this end, the use of MAs for autonomic data fusion tasks has been a subject of intense research during the past few years. However, several research issues remain open, as outlined below:

- SNs in a WSN typically operate unattended, and are therefore vulnerable to tampering. Hence, the capture of an MA by an adversary is relatively easy and its collected data can then easily be retrieved. Also the MA may acquire deceitful data by a compromised node. Research should therefore be directed to MA-based schemes that provide effective, low-complexity security mechanisms for the MAs and privacy for their carried data. Techniques that allow MAs to identify tampered and unreliable SNs should also be investigated.
- WSNs can be threatened by denial-of-service (DoS) attacks (e.g. jamming, interference or resource exhaustion) which can cause the collapse of an entire network. DoS attacks represent a serious concern, especially in the eye of sensitive WSN application scenarios (e.g. battlefield surveillance). Thus, MA-based data fusion schemes able of responding to DoS attacks without disrupting their data fusion tasks should be derived.

- WSNs often face topology changes (due to temporal communication problems or nodes energy exhaustion), especially when deployed in hostile environments. Hence, research on MA-based data fusion should propose methods that guarantee the best attainable results in these environments. These methods should allow the fast and ‘inexpensive’ (of low complexity) adaptation of agent itineraries to topology modifications so that the overall fusion cost does not increase considerably. The key issues to be investigated is ‘when’ and ‘how’ to perform the itinerary adaptation; also ‘who’ (the PE or the MAs) will execute the itinerary adaptation procedure.
- Proposal of innovative WSN agent-oriented applications (apart from data fusion) that will benefit from the distributed nature of mobile agent objects and their ability to perform local data processing and filtering.
- Extensive evaluation of mobile agent paradigm in a variety of DSNs applications, such as object monitoring and tracking.
- Development of formal theoretical models that will systematically analyse the qualitative and quantitative trade-offs of the mobile agent approach vs. the client-server model in WSN environments, in a way similar to the one followed for IP networks [14].

10 Conclusion

This chapter reviewed the main aspects of data fusion, MA technology and the benefits gained by utilizing MAs for autonomic data fusion tasks in WSNs.

It also classifies the research works that deal with MA-based autonomic data fusion in WSNs in five main categories: single MA-based, multiple MA-based, autonomic data fusion in clustered WSN architectures, hardware-based and combined multiple MA/stationary agents-based autonomic data fusion, highlighting their relevant merits and shortcomings. Furthermore, it highlights open research issues in the field of MA-based autonomic data fusion in WSNs. In the near future, the wider adoption and usage of WSN technologies is expected to bring out the significant role that MAs can play in this type of networks.

References

1. Akyildiz, F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks. *IEEE Communications Magazine*, pp. 102–114, August 2002.
2. Boulis, A.: Programming Sensor Networks with Mobile Agents. *Proceedings of the 6th International Conference on Mobile Data Management (MDM'2005)*, pp. 252–256, May 2005.
3. Boulis, A., Han, C., Srivastava, M.: Design and Implementation of a Framework for Efficient and Programmable Sensor Networks. *Proc. ACM MobiSys '03*, pp. 187–200, May 2003.
4. Chong, C.Y., Kumar, S.P.: Sensor networks: evolution, opportunities, and challenges. *Proceeding of the IEEE*, Vol. 91, NO. (8), 1247–1256, Aug. 2003.
5. Chen, M., Kwon, T., Choi, Y.: Data Dissemination based on Mobile Agent in Wireless Sensor Networks. *Proceedings of the 30th IEEE Conference on Local Computer Networks (LCN'05)*, pp. 527–529, Nov. 2005.

6. Chen, M., Kwon, T., Yuan, Y., Choi, Y., Leug, V.C.M.: Mobile Agent-Based Directed Diffusion in Wireless Sensor Networks. EURASIP Journal on Advances in Signal Processing Volume 2007, Hindawi Publishing Corporation, 2007.
7. Chu, M., Haussecker, H., Zhao, F.: Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High-Performance Computing Applications*, Vol. 16, No. 3, pp. 293–313 (2002).
8. Elen, B., Michiels, S., Joosen, W., Verbaeten, P.: A middleware pattern to support complex sensor network applications. OOPSLA '06, Workshop on building software for sensor networks, Portland, Oregon, USA, (2006).
9. Esau, L.R., Williams K.C.: On teleprocessing system design. Part II- A method for approximating the optimal network. *IBM Systems Journal*, 5, 142–147, 1966.
10. Esparza, O., Fernandez, M., Soriano, M., Munoz, J.L., Forne, J.: Mobile Agents Watermarking and Fingerprinting: Tracing Malicious Hosts. DEXA 2003, LNCS 2736, Springer-Verlag, (2003).
11. Fisher, L.: Protecting Integrity and Secrecy of Mobile Agents on Trusted and Non-Trusted Agent Places, diploma thesis, Department of computer science, University of Bremen, 2003.
12. Flinn, J., Satyanarayanan, M.: Energy-aware adaptation for mobile applications. *Symposium on Operating Systems Principles*, pp. 48–63, 1999.
13. Fok, C., Roman, G., Lu, C.: Mobile Agent Middleware for Sensor Networks: An Application Case Study. *Proc. IEEE IPSN '05*, pp. 382–387, Apr. 2005.
14. Fuggeta, A., Picco, G.P., Vigna, G.: Understanding Code Mobility. *IEEE Transactions on Software Engineering* 24(5), 346–361, 1998.
15. Gavalas, D.: Mobile Software Agents for Network Monitoring and Performance Management, PhD Thesis, University of Essex, UK, July 2001.
16. Gavalas, D., Pantziou, G., Konstantopoulos, C., Mamalis, B.: New Techniques for Incremental Data Fusion in Distributed Sensor Networks. In *Proceedings of the 11th Panhellenic Conference on Informatics (PCI'2007)*, pp. 599–608, (2007).
17. Geihs, K.: Middleware Challenges Ahead. *IEEE Computer*, pp. 24–31, (2001).
18. Al-Hammouri, A., Zhang, W., Buchheit, R., Liberatore, V., Chrysanthis, P., Pruhs, K.: Network Awareness and Application Adaptability. *Information Systems and E-Business Management*, 4(4), 399–419, Oct. 2006.
19. Iyengar, S. S., Jayasimha, D. N., Nadig, D.: A versatile architecture for the distributed sensor integration problem. *IEEE Trans. Comput.*, 43(2), pp. 175–185, Feb. 1994.
20. Guibas, L.J. Sensing, tracking, and reasoning with relations. *IEEE Signal Processing Magazine* 2002, pp. 73–85, 2002.
21. Heinzelman, W., Kulik, J., Balakrishnan, H.: Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of 5th ACM/IEEE Mobicom Conference*, pp. 174–185, (1999).
22. Heinzelman, W.B., Murphy, A.L., Carvalho, H.S., Perillo M.A.: Middleware to Support Sensor Network Applications. *IEEE Network*, pp. 6–14, Jan./Feb. 2004.
23. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. *Proceedings of the 6th Annual International conference on Mobile Computing and Networking*, 56–67, 2000.
24. Jayasimha, D.N., Iyengar, S.S., Kashyap, R.L.: Information integration and synchronization in distributed sensor networks. *IEEE Trans. Syst., Man, Cybern.*, 21(21), 1032–1043, Sept./Oct. 1991.
25. Jiao, Y., Hurson, A.R.: Adaptive Power Management for Mobile Agent-Based Information Retrieval. *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, pp. 675–680, Mar. 2005.
26. Kendall, E.A., Krishna, P.V.M., Pathak, C.V., Suresh, C.B.: Patterns of Intelligent and Mobile Agents. *Proceedings of the 2nd International Conference on Autonomous Agents (Agents98)*, pp. 92–99, May 1998.
27. Kershenbaum, A.: *Telecommunications Network Design Algorithms*, McGraw-Hill, 1993.

28. Knoll, A., Meinkoehn, J.: Data fusion using large multi-agent networks: an analysis of network structure and performance. In Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Las Vegas, NV, IEEE, pp. 113–120, Oct. 2–5 1994.
29. Krause, S., Magedanz, T.: Mobile Service Agents enabling Intelligence on Demand in Telecommunications. Proceedings of IEEE GLOBECOM 96, pp. 78–84, Nov. 1996.
30. Lange, D.B., Oshima, M.: Seven Good Reasons for Mobile Agents. Communications of the ACM, 42(3), 88–89, Mar. 1999.
31. Levis, P., Culler, D.: Mat?: a tiny virtual machine for sensor networks. In ASPLOS- X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, pp. 85–95, 2002.
32. Li, S., Son, S.H., Stankovic, J.A.: Event detection services using data service middleware in distributed sensor networks. In IPSN'03: Proceedings of the 2nd international symposium on Information processing in sensor networks, pp. 502–517, 2003.
33. Lindsey, S., Raghavendra, C. S.: PEGASIS: Power Efficient GATHERing in Sensor Information Systems. In Proceedings of IEEE Aerospace Conference, vol.3, pp. 1125–1130, 2002.
34. Lindsey, S., Raghavendra, C.S., Sivalingam, K.: Data Gathering in Sensor Networks using the Energy*Delay Metric. In Proceedings of the IPDPS Workshop on Issues in Wireless Networks and Mobile Computing, pp. 2001–2008, 2001.
35. Liu, T., Martonosi, M.: Impala: a middleware system for managing autonomic, parallel sensor systems, In PPOPP'03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp.107–118, 2003.
36. Lotfinezhad, M., Liang, B.: Energy Efficient Clustering in Sensor Networks with Mobile Agents. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'05), vol. 3, pp. 1872–1877, Mar. 2005.
37. Madden, S., Franklin, M.J., Hellerstein, J.M.: TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. In Proceedings of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI'02), vol. 36, pp. 131–146, USENIX, Dec. 2002.
38. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In SIGMOD'03: proceedings of the ACM SIGMOD international conference on management of data, pp. 491–502, 2003.
39. Malik, H., Shakshuki, E., Dewolf, T.: Multi-agent System for Directed Diffusion in Wireless Sensor Networks. 21st International Conference on Advanced Information Networking and Applications Workshops, Niagara Falls, ON, Canada, vol. 2, pp. 635–640, May 2007.
40. Milojevic D.: Mobile Agent Applications. IEEE Concurrency, 7(3), July–Sep. 1999.
41. Mpitziopoulos, A., Gavalas, D., Konstantopoulos, C., Pantziou, G.: Deriving efficient mobile agent routes in wireless sensor networks with NOID algorithm. In Proceedings of the 18th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'2007), Athens, Greece, pp. 1–5, Sep. 2007.
42. Pham, V., Karmouch, A.: Mobile Software Agents: An Overview. IEEE Communications Magazine, 36(7), pp. 26–37, (1998).
43. Pieprzyk, J., Hardjono, T., Seberry, J.: Fundamentals of Computer Security. Springer-Verlag, Berlin, 2003.
44. Prasad, L., Iyengar, S.S., Rao, R.L.: Fault-tolerant sensor integration using multiresolution decomposition. Physical Review E, 49(4), 3452–3461, Apr. 1994.
45. Qi, H., Wang, F.: Optimal Itinerary Analysis for Mobile Agents in Ad Hoc Wireless Sensor Networks. Proceedings of the 13th International Conference on Wireless Communications (Wireless'2001), pp. 147–153, 2001.
46. Qi, H., Iyengar, S.S., Chakrabarty, K.: Multi-resolution data integration using mobile agents in distributed sensor networks, IEEE Trans. on Syst., Man, and Cybern. Part C: Applications and Reviews 31(3), 383–391, 2001.
47. Qi, H., Kurganti, P. T., Xu, Y.: The Development of Localized Algorithms in Wireless Sensor Networks. Sensors, 2, 286–293, 2002.

48. Raghunathan, V., Schurgers, C., Park, S., Srivastava, M.B.: Energy-aware wireless microsensor networks, *IEEE Signal Processing Magazine* 19(2), 40–50, 2002.
49. Reuter, E., Baude, F.: System and Network Management Itineraries for Mobile Agents. Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications (MATA'02), LNCS vol. 2521, pp. 227–238, Oct. 2002.
50. Rubinstein, M.G., Duarte, O.C., Pujolle, G.: Scalability of a Mobile Agents Based Network Management Application. *Journal of Communications and Networks*, 5(3), Sept. 2003.
51. Sander, T., Tschudin, Ch. F.: Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, vol. 1419 of LNCS. Springer-Verlag, 1998.
52. Savvides, A., Han, C.C., Srivastava, M.B.: Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proc. Seventh Ann. ACM/IEEE Int. Conf. on Mobile Computing and Networking (Mobicom 2001)*, ACM press, Rome, Italy, pp. 166–179, 2001.
53. Shih, D.H., Huang, S.Y., Yen, D.C.: A New Reverse Auction Agent System for m-Commerce Using Mobile Agents, *Computer Standards & Interfaces*, 27(4), 383–395, Apr. 2005.
54. Silva, F., Heidemann, J., Govindan, R., Estrin, D.: Directed Diffusion, Technical Report ISI-TR-2004-586, USC/Information Sciences Institute, Jan. 2004.
55. Tong, L., Zhao, Q., Adireddy, S.: Sensor Networks with Mobile Agents. In *Proc. IEEE MIL-COM'03*, Boston, MA, pp. 1–6, Oct. 2003.
56. Tseng, Y.C., Kuo, S.P., Lee, H.W., Huang, C.F.: Location Tracking in a Wireless Sensor Network by Mobile Agents and Its Data Fusion Strategies. *Computer Journal*, 47(4), 448–460, 2004.
57. Tiny Os: <http://www.tinyos.net/>. Cited 20 Feb. 2008.
58. Umezawa, U., Satoh, I., Anzai, Y.: A Mobile Agent-Based Framework for Configurable Sensor Networks, Proceedings of the 4th International Workshop on Mobile Agents for Telecommunications Applications (MATA'02), pp. 128–140, Oct. 2002.
59. Vigna, G.: Protecting Mobile Agents through Tracing. Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Finland, June 1997.
60. Vigna, G.: Cryptographic traces for mobile agents. In *Mobile Agents and Security*, vol. 1419 of LNCS. Springer-Verlag, 1998.
61. Xu, Y., Qi, H.: Distributed Computing Paradigms for Collaborative Signal and Information Processing in Sensor Networks. *Int'l. J. Parallel and Distrib. Comp.*, 64(8), 945–959, Aug. 2004.
62. Wald L.: Some terms of reference in data fusion. *IEEE Transactions on Geosciences and Remote Sensing*, 37(3), 1190–1193, 1999.
63. Wang, X., Bi, D.W., Ding, L., Wang, S.: Agent Collaborative Target Localization and Classification in Wireless Sensor Networks. *Sensors* 7, 1359–1387, 2007.
64. Wu, Q., Rao, N., Barhen, J., Iyengar, S., Vaishnavi, V., Qi, H., Chakrabarty, K.: On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks. *IEEE Transactions on Knowledge and Data Engineering*, 16(6), 740–753, June 2004.
65. Zaslavsky, A.: Mobile Agents: Can They Assist with Context Awareness? 2004 IEEE MDM'04, Berkeley, California, pp. 304–305, Jan. 2004.
66. Zhao, F., Shin, J., Reich, J.: Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2), 61–72, Mar. 2002.
67. Zhao, F., Guibas, L.: *Wireless Sensor Networks*. Morgan Kaufmann Publishers, San Francisco, 2004.

Component-Based Autonomic Management for Legacy Software

Daniel Hagimont, Patricia Stolf, Laurent Broto, and Noel De Palma

Abstract Distributed software environments are increasingly complex and difficult to manage, as they integrate various legacy software with specific management interfaces. Moreover, the fact that management tasks are performed by humans leads to many configuration errors and low reactivity. This is particularly true in medium or large-scale distributed infrastructures. To address this issue, we explore the design and implementation of an autonomic management system. The main principle is to wrap legacy software pieces in components in order to administrate a software infrastructure as a component architecture. In order to help the administrators defining autonomic management policies, we introduce high-level formalisms for the specification of deployment and management policies. This chapter describes the design and implementation of such a system and its evaluation with different use cases.

1 Introduction

Today's computing environments are becoming increasingly sophisticated. They involve numerous complex software that cooperate in potentially large-scale distributed environments. These software are developed with very heterogeneous programming models and their configuration facilities are generally proprietary. Therefore, the management of these software (installation, configuration, tuning, repair, etc.) is a much complex task which consumes a lot of resources:

- human resources as administrators have to react to events (such as failures) and have to reconfigure (repair) complex applications,
- hardware resources which are often reserved (and overbooked) to anticipate load peaks or failures.

D. Hagimont (✉)
INPT, Toulouse, France
e-mail: hagimont@enseeiht.fr

A very promising approach to the above issue is to implement administration as an autonomic software. Such a software can be used to deploy and configure applications in a distributed environment. It can also monitor the environment and react to events such as failures or overloads and reconfigure applications accordingly and autonomously. The main advantages of this approach are as follows:

- Providing a high-level support for deploying and configuring applications reduces errors and administrators efforts.
- Autonomic management allows the required reconfigurations to be performed without human intervention, thus saving administrators time.

In order to provide such an autonomic administration software, we rely on the concepts of components and software architectures. The main principle is to wrap legacy software pieces in components in order to administrate a software infrastructure as a component architecture. As soon as a legacy software has been wrapped in a component, the administrator can describe a software environment to deploy using the component model's ADL (Architecture Description Language) and implement reconfiguration programs (autonomic managers) using the component model's APIs. A prototype (called Jade [14]) of such a component-based management system has been implemented and used for the administration of different applications.

However, we observed that the interfaces of a component model are too low level and difficult to use. In order to implement wrappers (to encapsulate existing software), to describe deployed architectures and to implement reconfiguration programs, the administrator of the environment has to learn (yet) another framework, the Fractal [7] component model in the case of Jade. Tune is an evolution of Jade which aims at providing a higher level formalism for all these tasks (wrapping, deployment, reconfiguration). The main motivation is to hide the details of the component model we rely on and to provide a more intuitive policy specification interface for wrapping, deployment and reconfiguration.

This chapter presents our experience in designing, implementing and using these two prototypes. In the following section (Section 2), we present the two use cases that we used to illustrate and evaluate our prototypes. Section 3 presents the design of the Jade system which implements the component-based autonomic management approach. Section 4 presents the design of Tune, which relies on Jade and provides high-level language support for the definition of administration policies. Section 5 presents scenarios and evaluations we conducted with the previous platform and use cases. After a presentation of related works in Section 6, we conclude in Section 7.

2 Use Cases

Our main application target is the administration of servers distributed over a cluster of machines or a grid infrastructure. We give two examples of such organizations.

2.1 Multi-Tier Internet Services

The first experimental environment we consider is the Java 2 Platform, Enterprise Edition (J2EE), which defines a model for developing web applications [27] in a multi-tiered architecture. Such applications usually receive requests from web clients, that flow through a web server (provider of static content), then to an application server to execute the business logic of the application and generate web pages on-the-fly, and finally to a database that persistently stores data (see Fig. 1).

Upon an HTTP client request, either the request targets a static web document, in which case the web server directly returns that document to the client; or the request refers to a dynamic document, in which case the web server forwards that request to the application server. When the application server receives a request, it runs one or more software components (e.g. Servlets, EJBs) that query a database through a JDBC (Java DataBase Connection) driver [28]. Finally, the resulting information is used to generate a web document on-the-fly that is returned to the web client.

In this context, the increasing number of Internet users has led to the need of highly scalable and highly available services. Moreover, several studies show that the complexity of multi-tier architectures with their dynamically generated documents represent a large portion of web requests, and that the rate at which dynamic documents are delivered is often one or two orders of magnitudes slower than static documents [15, 16]. This places a significant burden on servers [10]. To face high loads and provide higher scalability of Internet services, a commonly used approach is the replication of servers in clusters. Such an approach usually defines a particular (hardware or software) component in front of the cluster of replicated servers, which dynamically balances the load among the replicas. Here, different load balancing algorithms may be used, e.g. Random, Round-Robin. Among the existing J2EE clustering solutions we can cite C-JDBC for a cluster of database servers [11], JBoss clustering for a cluster of JBoss EJB servers [8], mod_jk for a cluster of Tomcat Servlet servers [24], and the L4 switch for a cluster of replicated Apache web servers [26] (see Fig. 2).

Clustered multi-tier J2EE systems represent an interesting experimental environment for our autonomic management environment, since they bring together all the addressed challenges:

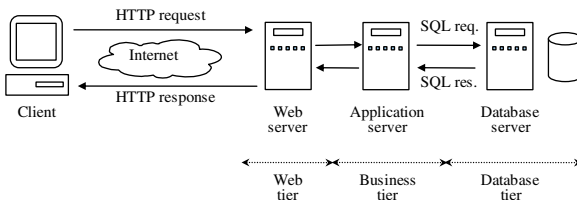


Fig. 1 Architecture of dynamic web applications

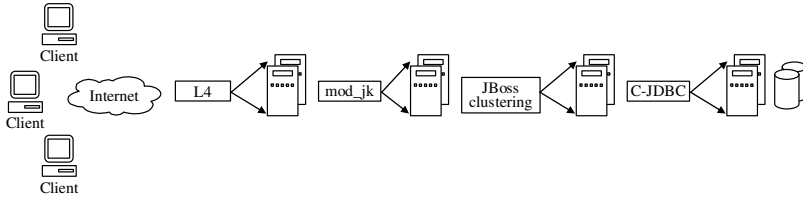


Fig. 2 J2EE clustering

- the management of a variety of legacy systems, since each tier in the J2EE architecture embeds a different piece of software (e.g. a web server, an application server or a database server),
- very complex administration interfaces and procedures associated with very heterogeneous software,
- the requirement for high reactivity in taking into account events which may compromise the normal behaviour of the managed system, e.g. load peaks or failures.

2.2 DIET : Distributed Load Balancer Over a Grid

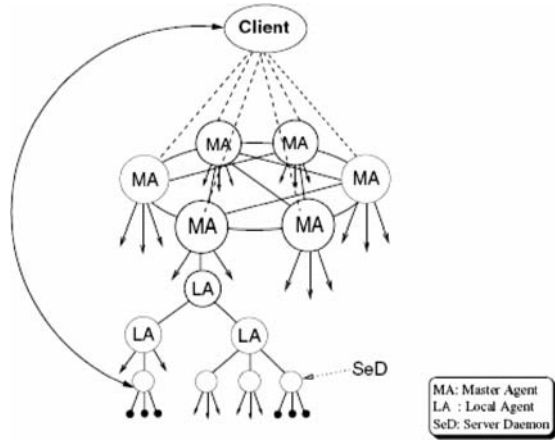
Grid computing aims at enabling the sharing, selection and aggregation of geographically distributed resources, dynamically at runtime depending on their availability, capability, cost and user's quality of service (QoS) requirements [9]. Diet [12] is an example of middleware environment which aims at balancing computation load over a grid. It is built on top of different tools which are able to locate an appropriate server depending on the client requested function, the data location (which can be anywhere on the system, because of previous computations) and the dynamic performance characteristics of the system. The aim of Diet is to provide transparent access to a pool of computational servers at a very large scale.

As illustrated in Fig. 3, Diet mainly has the following components. A client is an application which uses DIET to solve problems. A Master Agents (MA) receive computation requests from clients. Then a MA chooses the best server and returns its reference to the client. The client then sends the computation request to that server. Local agents (LA) aim at transmitting monitoring information between servers and MAs. LAs do not take scheduling decision, but allow preventing MAs overloads when dealing with large-scale infrastructures. Server Daemons (SeDs) encapsulate computational servers (processors or clusters). A SeD declares the problems it can solve to its parent LA and provides an interface to clients for submitting their requests.

The DIET middleware also represents an interesting example for autonomic management:

- the management of a distributed organization of legacy software (MAs, LAs and SeDs),
- distributed configuration, since DIET requires the configuration of these software (through a set of configuration files) to implement a consistent hierarchical structure,

Fig. 3 The Diet distributed load balancer. Diet is organized as a tree structure, where each MA is linked with a set of LA. Each LA can be linked with a set of LA, or a set of SeD which are the leaves of the tree and include the computing programs



- self-repair: due to hardware or software failures, a server may stop functioning. The goal is to detect the failure and repair it automatically.

In both examples (J2EE and Diet), the deployment of the servers in a cluster or grid is very complex and requires a lot of expertise. Many files have to be edited and configured consistently. Also, failures or load peaks (when the chosen degree of replication is too low) must be treated manually.

3 Component-Based Autonomic Management

Component-based management aims at providing a uniform view of a software environment composed of different types of servers. Each managed server is encapsulated in a component and the software environment is abstracted as a component architecture. Therefore, deploying, configuring and reconfiguring the software environment is achieved by using the tools associated with the used component-based middleware. This solution is followed by several research projects [13, 14, 21] including Jade.

The component model we used in Jade is the Fractal component model [7] which is described in the next section.

3.1 The Fractal Component Model

The Fractal component model is a general component model which is intended to implement, deploy, monitor and dynamically configure, complex software systems, including in particular operating systems and middleware. This motivates the main features of the model: composite components (to have a uniform view of applications at various levels of abstraction), introspection capabilities (to monitor, and control the execution of a running system) and re-configuration capabilities (to deploy, and dynamically configure a system).

A Fractal component is a runtime entity that is encapsulated, and that has a distinct identity. A component has one or more interfaces. An interface is an access point to a component, that supports a finite set of methods. Interfaces can be of two kinds: server interfaces, which correspond to access points accepting incoming method calls, and client interfaces, which correspond to access points supporting outgoing method calls. The signatures of both kinds of interface can be described by a standard Java interface declaration, with an additional role indication (server or client). A Fractal component can be composite, i.e. defined as an assembly of several sub-components, or primitive, i.e. encapsulating an executable program.

Communication between Fractal components is only possible if their interfaces are bound. Fractal supports both primitive bindings and composite bindings. A primitive binding is a binding between one client interface and one server interface in the same address space. A composite binding is a Fractal component that embodies a communication path between an arbitrary number of component interfaces. These bindings are built out of a set of primitive bindings and binding components (stubs, skeletons, adapters, etc).

The above features (hierarchical components, explicit bindings between components, strict separation between component interfaces and component implementation) are relatively classical. The originality of the Fractal model lies in its open reflective features. In order to allow for well-scoped dynamic reconfiguration, components in Fractal can be endowed with controllers, which provide access to a component internals, allowing for component introspection and the control of component behaviour.

A controller provides a control interface and implements a control behaviour for the component, such as controlling the activities in the components (suspend, resume) or modifying some of its attributes. The Fractal model allows for arbitrary (including user defined) classes of controller. It specifies, however, several useful forms of controllers, which can be combined and extended to yield components with different control features, including the following:

- Attribute controller: an attribute is a configurable property of a component. This controller supports an interface to expose getter and setter methods for its attributes.
- Binding controller: supports an interface to allow binding and unbinding its client interfaces to server interfaces.
- Content controller: for composite components, supports an interface to list, add and remove subcomponents in its contents.
- Life-cycle controller: this controller allows an explicit control over a component execution. Its associated interface includes methods to start and stop the execution of the component.

An ADL (XML-based language) allows describing an architecture and an ADL launcher can be used to deploy such an architecture.

Several implementations of the Fractal model have been issued in different contexts, e.g. an implementation devoted to the configuration of operating systems on a bare hardware (Think) or an implementation on top of the Java virtual

machine (Julia) targeted to the configuration of middleware or applications. The work reported in this chapter relies on this later implementation of Fractal on top of Java.

3.2 Component-Based Management

Any software managed with Jade is wrapped in a Fractal component which interfaces its administration procedures. Therefore, the Fractal component model is used to implement a management layer (Fig. 4) on top of the legacy layer (composed of the actual managed software). In the management layer, all components provide a management interface for the encapsulated software, and the corresponding implementation (the wrapper) is specific to each software (e.g. the Apache web server in the case of J2EE or the Master Agent in the case of Diet). Fractal’s control interfaces allow managing the element’s attributes and bindings with other elements, and the management interface of each component allows controlling its internal configuration state. Relying on this management layer, sophisticated administration programs can be implemented, without having to deal with complex, proprietary configuration interfaces, which are hidden in the wrappers.

The above approach is illustrated in Fig. 4 in the case of a J2EE architecture. In this setting, an L5-switch balances the requests between two Apache server replicas. The Apache servers are connected to two Tomcat server replicas. The Tomcat servers are both connected to the same MySQL server. The vertical arrows (between the management and legacy layers) represent management relationships between components and the wrapped software entities. In the legacy layer, the dashed lines represent relationships (or bindings) between legacy entities, whose implementations are proprietary. These bindings are represented in the management layer by (Fractal) component bindings (full lines in the figure).

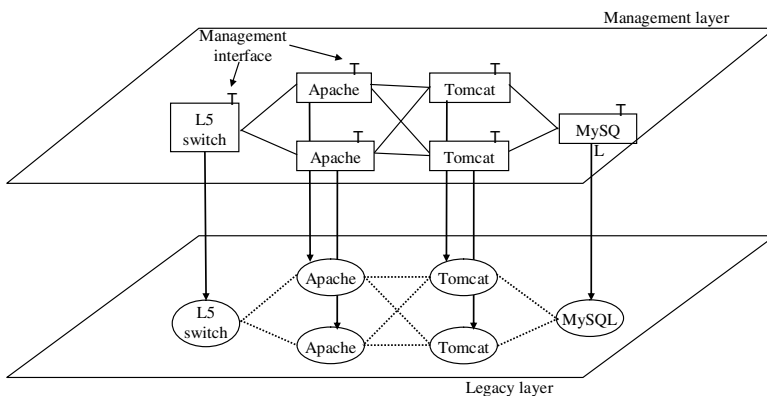


Fig. 4 Implementation of a management layer on top of the legacy software layer

We now give an example of a Fractal wrapper for the Apache server that is part of the J2EE architecture. The wrapper provides an attribute controller, a binding controller and a lifecycle controller:

- The attribute controller interface is used to set attributes related to the local execution of the Apache server. For instance, a modification of the port attribute of the Apache component is reflected in the *httpd.conf* file in which the port attribute is defined.
- The binding controller interface is used to connect Apache with other middleware tiers. For instance, invoking the bind operation on the Apache component sets up a binding between one instance of Apache and one instance of Tomcat. The implementation of this bind method is reflected at the legacy layer in the *worker.properties* file used to configure the connections between Apache and Tomcat servers.
- The life cycle controller interface is used to start or to stop the server as well as to read its state (i.e. running or stopped). It is implemented by calling the Apache commands for starting/stopping a server.

Other servers (Tomcat and MySQL) are wrapped in a similar way into Fractal components, and provide the same management interface.

Here, we distinguish two important roles:

- the role of the management and control interfaces is to provide a means for configuring components and bindings between components. It includes methods for navigating in the component-based management layer or modifying it to implement reconfigurations.
- the role of the wrappers is to reflect changes in the management layer onto the legacy layer. The implementation of a wrapper for a specific software may also have to navigate in the component management layer, to access key attributes of the components and generate legacy software configuration files. For instance, the configuration of an Apache server requires to know the name and location of the Tomcat servers it is bound to.

3.3 Deployment

The architecture of an application is described using an ADL, which is one of the basic features of the Fractal component model. This description is an XML document which details the architectural structure of the application to deploy, e.g. which software resources compose the multi-tier J2EE application, how many replicas are created for each tier and how are the tiers bound together.

A Software Installation Service component (a component of Jade) allows retrieving the encapsulated software resources involved in the application (e.g., Apache Web server software, MySQL database server software) and installing them on nodes. A Cluster Manager component is responsible for the allocation of nodes (from a pool of available nodes) which will host the deployed software elements.

The deployment of an application is the interpretation of an ADL description, using the Software Installation Service and the Cluster Manager to deploy application’s components on nodes.

3.4 Implementing Autonomic Managers

Autonomic computing is achieved through autonomic managers, which implement feedback control loops. These loops regulate and optimize the behaviour of the managed system. Figure 5 illustrates control loops in the Jade autonomic management system. It shows two managers that regulate two specific aspects of the platform (self-recovery and self-optimization). Each autonomic manager in Jade is based on a control loop that includes sensor, actuator and analysis/decision components.

Sensors are responsible for the detection of the occurrence of a particular event, e.g. a QoS requirement violation in case of a self-optimization manager, or an element failure (node, middleware or component) for a self-recovery manager.

Analysis/decision components (or reactors) represent the actual reconfiguration algorithm, e.g. repairing a failed element in case of a self-recovery manager, or resizing the cluster of replicated servers upon load changes in case of a self-optimization manager. Reactors receive notifications from sensors and make use of actuators when a reconfiguration operation is necessary.

Actuators represent the individual mechanisms necessary to implement reconfiguration operations, e.g. allocating a new node to a cluster of replicas, adding/removing a replica to the cluster of replicated servers, updating connections between the tiers.

Sensors, Actuators and Reactors are implemented as Fractal components, which allows reusing and combining them to assemble specific autonomic managers.

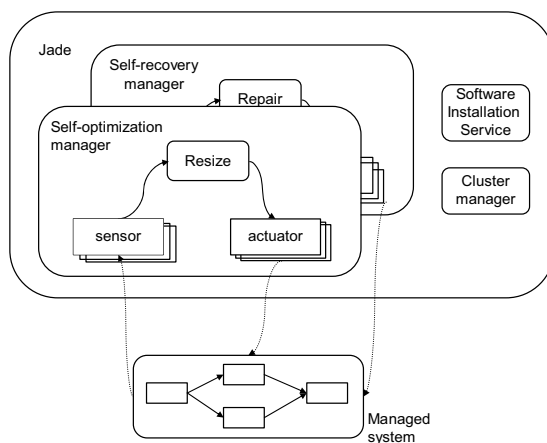


Fig. 5 Control loops

Component-based autonomic computing has proved to be a very convenient approach. The experiments we conducted with Jade for managing J2EE or Diet infrastructures validated this design choice (see Section 5). In the Jade system, an administrator can wrap legacy software in components, describe a software environment to deploy using the component model ADL and implement reconfiguration programs (autonomic managers) using the component model's interfaces (Java interfaces in Fractal).

4 Autonomic Management Policies Specification

As Jade was used by external users (external to our group), we observed that:

- wrapping components are difficult to implement. The developer needs to have a good understanding of the component model we use (Fractal),
- deployment is not very easy. ADLs are generally very verbose and still require a good understanding of the underlying component model. Moreover, if we consider large-scale software infrastructure such as those deployed over a grid (as in the Diet example), deploying a thousand of servers requires an ADL deployment description file of several thousands of lines,
- autonomic managers (reconfiguration policies) are difficult to implement as they have to be programmed using the management and control interfaces of the management layer. This also required a strong expertise regarding the used component model.

All these observations led us to the conclusion that a higher level interface was required for describing the encapsulation of software in components, the deployment of a software environment potentially in large scale and the reconfiguration policies to be applied autonomically. Tune is an evolution of Jade which aims at providing a higher level formalism for all these tasks (wrapping, deployment, reconfiguration). The main motivation is to hide the details of the component model we rely on and to provide a more intuitive policy specification interface for wrapping, deployment and reconfiguration.

4.1 *Tune's Management Interface*

As previously motivated, our goal is to provide a high-level interface for the description of the application to wrap, deploy and reconfigure. This led us to the following design choices:

- Regarding wrapping, our approach is to introduce a Wrapping Description Language (WDL) which is used to specify the behaviour of wrappers. A WDL specification is interpreted by a generic wrapper Fractal component, the specification and the interpreter implementing an equivalent wrapper. Therefore, an administrator does not have to program any implementation of Fractal component.

- Regarding deployment, our approach is to introduce a Unified Modelling Language (UML) profile for graphically describing deployment schemas. First, a UML-based graphical description of such a schema is much more intuitive than an ADL specification, as it does not require expertise of the underlying component model. Second, the introduced deployment schema is more abstract than the previous ADL specification, as it describes the general organization of the deployment (types of software to deploy, interconnection pattern) in intension, instead of describing in extension all the software instances that have to be deployed. This is particularly interesting for applications like Diet where thousands of servers have to be deployed.
- Regarding reconfiguration, our approach is to introduce a UML profile for the description of state diagrams. These state diagrams are used to define workflows of operations that have to be performed for reconfiguring the managed environment. One of the main advantage of this approach, besides simplicity, is that state diagrams manipulate the entities described in the deployment schema and reconfigurations can only produce an (concrete) architecture which conforms with the abstract schema, thus enforcing reconfiguration correctness.

All these policy specifications (wrapping, deployment and reconfiguration) are interpreted by a runtime middleware which deploys and autonomously manages the application according to the policies. We detail these three aspects in the next sections.

4.2 A UML Profile for Deployment Schemas

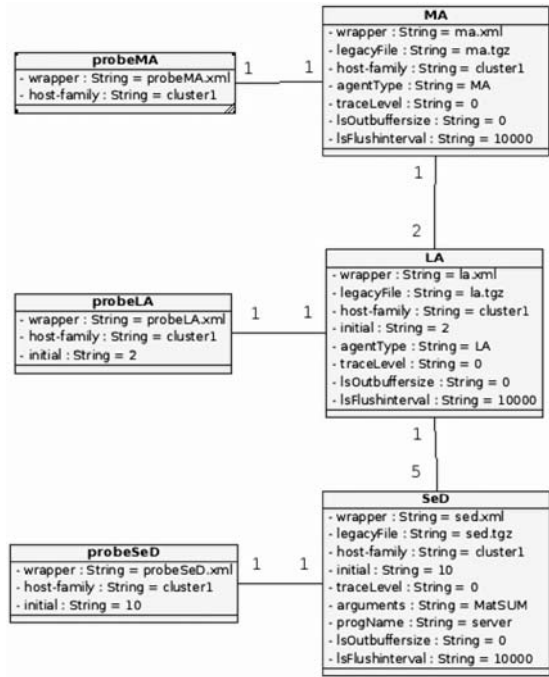
The UML profile we introduce for specifying deployment schemas is illustrated in Fig. 6 where a deployment schema is defined for a Diet organization. A deployment schema describes the overall organization of a software infrastructure to be deployed. At deployment time, the schema is interpreted to deploy a component architecture. Each element (the boxes) corresponds to a software which can be instantiated in several component replicas. A link between two elements generates bindings between the components instantiated from these elements. Each binding between two components is bi-directional (actually implemented by two bindings in opposite directions), which allows navigation in the component architecture.¹

An element includes a set of configuration attributes for the software (all of type String). Most of these attributes are specific to the software, but few attributes are predefined by Tune and used for deployment:

- **wrapper** is an attribute which gives the file name of the WDL description of the software's wrapper,

¹ We chose to implement bi-directional bindings in order to allow arbitrary navigation in the component architecture, although this is not actually mandatory.

Fig. 6 Deployment schema for Diet. The schema describes a Diet organization where one MA, two LAs and 10 SeDs (5 for each LA) should be deployed. A probe is linked with each software, which monitors the liveness of the server in order to trigger a repair procedure. This schema deploys a component architecture as illustrated in Fig. 3



- **legacyFile** is an attribute which gives the file name of the archive which contains the legacy software binaries and configuration files,
- **hostFamily** is an attribute which gives a hint regarding the dynamic allocation of the nodes where the software should be deployed,
- **initial** is an attribute which gives the number of instances which should be deployed. The default value is 1.

A cardinality is associated with each link. If $A(n)$ and $B(m)$ are two linked elements in a schema, with an initial attribute (initial number of instances) n for A and m for B , the semantic of the cardinality is the following. A link $A(n) t - u B(m)$ means that each A component should be bound with $u B$ components and each B component should be bound with $t A$ components. The cardinality is constrained by $m = nu/t$ with $m \geq u$ and $n \geq t$.

One of the main benefit of the introduced formalism is that it allows describing in a synthetic schema the deployment of several hundreds of servers. Clustered J2EE architectures can also easily be described thanks to this graphical language.

4.3 A Wrapping Description Language

Upon deployment, the above schema is parsed and for each element, a number of Fractal components are created. These Fractal components implement the wrappers for the deployed software, which provide control over the software. Each wrapper

Fractal component is an instance of a generic wrapper which is actually an interpreter of a WDL specification.

A WDL description defines a set of methods that can be invoked to configure or reconfigure the wrapped software. The workflow of methods that have to be invoked in order to configure and reconfigure the overall software environment is defined thanks to an interface introduced in Section 4.4.

Generally, a WDL specification provides *start* and *stop* operations for controlling the activity of the software, and a *configure* operation for reflecting the values of the attributes (defined in the UML deployment schema) in the configuration files of the software. Note that the values of these attributes can be modified dynamically. Other operations can be defined according to the specific management requirements of the wrapped software, these methods are implemented in Java.

The main motivation for the introduction of WDL are as follows:

- to hide the complexity of the underlying component model (Fractal),
- that most of the needs should be met with a finite set of generic methods (that can be therefore reused).

Figure 7 shows an example of WDL specification which wraps a SeD computing server in a Diet architecture. It defines *start* and *stop* methods which can be invoked to launch/stop the deployed SeD software, and a *configure* method which reflects configuration attributes in the configuration file of the SeD software. The Java implementations of these methods are generic and have been used in the wrappers of most of the software we wrapped (LA, MA for Diet, but also Apache, Tomcat and MySQL for J2EE. We only had to add an implementation of a *configure* method for XML configuration files). A method definition includes the description of the parameters that should be passed when the method is invoked. These parameters may be String constants, attribute values or combination of both (String expressions). All the attributes defined in the deployment schema can be used to pass the configured attributes as parameters of the method invocations. However, several additional attributes are automatically added and managed by Tune:

- *dirLocal* is the directory where the software is actually deployed on the target machine,
- *compName* is a unique name associated with the deployed component instance,
- *PID* is the process identifier of the process that runs the software.

In Fig. 7, the *start* method takes as parameters the shell command that launch the server, and the environment variables that should be set:

- $\$dirLocal/\$progName$ is the name of the binary to be launched (*progName* is an attribute of the wrapped software),
- $\$dirLocal/\$compName-cfg$ is the name of the configuration file which is passed to the binary and which is generated by the *configure* method of the wrapper $\$arguments$ is a parameter for the binary (also a software attribute),
- $LD_LIBRARY_PATH=\$dirLocal$ is an environment variable to pass to the binary.

```

<?xml version='1.0' encoding='ISO-8859-1' ?>
<wrapper name='sed'>
  <method name="start"
    key="appli.wrapper.util.GenericStart"
    method="start_with_pid_linux" >
    <param value="$dirLocal/$progName
      $dirLocal/$compName-cfg $arguments"/>
    <param value="LD_LIBRARY_PATH=$dirLocal"/>
  </method>

  <method name="configure"
    key="appli.wrapper.util.ConfigurePlainText"
    method="configure">
    <param value="$dirLocal/$compName-cfg"/>
    <param value=" = "/>
    <param value="traceLevel:$traceLevel" />
    <param value="parentName:$LA.compName"/>
    <param value="name:$compName"/>
    <param value="lsOutbuffersize:$lsOutbuffersize"/>
    <param value="lsFlushinterval:$lsFlushinterval"/>
  </method>

  <method name="stop"
    key="appli.wrapper.util.GenericStop"
    method="stop_with_pid_linux" >
    <param value="$PID"/>
  </method>
</wrapper>

```

Fig. 7 A WDL specification

The *configure* method is implemented by the *ConfigurePlainText* Java class. This configuration method generates a configuration file composed of <attribute,value> pairs:

- *\$dirLocal/\$compName-cfg* is the name of the configuration file to generate,
- = is the separator between each attribute and value,
- and the attributes and value are separated by a “:” character.

As evocated in Section 3.2, it is sometimes necessary to navigate in the deployed component architecture in order to configure the software. In the Diet example, a LA has a configuration variable (in its configuration file) called *Name* which is a unique name associated with the launched server. This configuration variable is assigned with the *\$compName* in its wrapper. A SeD which is a child of the LA must have a *parentName* configuration variable set to the name of the parent LA in its configuration file. Therefore, in the SeD wrapper (Fig. 7), we need to access the *compName* of its parent LA in order to set this *parentName* configuration variable. Since in the deployment schema there is a link between the LA and SeD elements, there are bindings between the LA and the SeDs at the component level. These bindings allow navigating in the management layer. In Fig. 7, the *parentName* configuration

variable is assigned with the name of the LA component which the SeD is bound with.

All the constructions (attributes, methods and navigation clauses) introduced in a WDL specification would be present in the equivalent Fractal component, but programmed with Fractal's API. In average, a Fractal wrapper in Jade represents 500 lines of Java code while the equivalent WDL specification represents 70 lines of XML (invoking reused Java libraries for configuring files or launching processes).

4.4 A UML Profile for (Re)configuration Procedures

Reconfigurations are triggered by events. An event can be generated by a specific monitoring component (e.g. probes in the deployment schema) or by a wrapped legacy software which already includes its own monitoring functions.

Whenever a wrapper component is instantiated, a communication pipe is created (typically a Unix pipe) that can be used by the wrapped legacy software to generate an event, following a specified syntax which allows for parameter passing. Note that the use of pipes allows any software (implemented in any language environment such as Java or C++) to generate events. An event generated in the pipe associated with the wrapper is transmitted to the administration node where it can trigger the execution of reconfiguration programs (in our current prototype, the administration code, which initiates deployment and reconfiguration, is executed on one administration node, while the administrated software is managed on distributed hosts). An event is defined as an event type, the name of the component which generated the event and possibly an argument (all of type String).

For the definition of reactions to events, we introduced a UML profile which allows specifying reconfiguration as state diagrams. Such a state diagram defines the workflow of operations that must be applied in reaction to an event. An operation in a state diagram can assign an attribute or a set of attributes of components, or invokes a method or a set of methods of components. To designate the components on which the operations should be performed, the syntax of the operations in the state diagrams allows navigation in the component architecture, similarly to the wrapping language.

For example, let us consider the diagram in Fig. 8 (on the left) which is the reaction to a LA (software) failure in Diet. The event (fixLA) is generated by a probeLA component instance, therefore the variable is the name of this probeLA component instance. Then:

- *this.stop* will invoke the *stop* method on the probing component (to prevent the generation of multiple events),
- *this.LA.start* will invoke the *start* method on the LA component instance which is linked with the probe. This is the actual repair of the faulting LA server,
- *this.LA.SeD.stop* will invoke the *stop* method on all the SeD component instances which are linked with this LA. This is necessary as in Diet, a restart of a LA

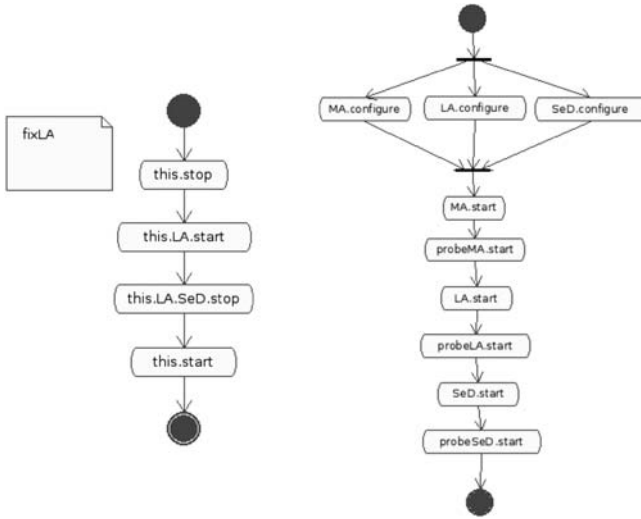


Fig. 8 State diagrams for repair and start

requires to restart all its SeD children in order to reconnect to the LA. Here, the probes associated with the SeDs will trigger the restart of the SeDs,

- *this.start* will restart the probe associated with the LA.

Note that state diagram’s operations are expressed using the elements defined in the deployment schema, and are applied on the actually deployed component architecture. In its latest version, Tune also provides operations which re-deploy components (change location or add component instances) while enforcing the defined abstract deployment schema.

A similar diagram is used to start the deployed Diet environment, as illustrated in Fig. 8 (on the right). In this diagram, when an expression starts with the name of an element in the deployment schema (LA or SeD, etc.), the semantic is to consider all the instances of the element, which may result in multiple method invocations. The starting diagram ensures that (1) configuration files must be generated, then (2) the servers must be started following the order MA, LA and SeDs. For each type of server, the server is started before its probe.

5 Evaluation

The application cases described in Section 2 have been implemented and Jade/Tune were used to deploy and administrate them.² The evaluations reported in this chapter focus on self-optimization for the database tier of a J2EE application and self-repair

² Both Jade and Tune are not intrusive (i.e. they do not intercept any message), and they only differ regarding the language support provided for management policy definition. So the evaluations below are valid for both systems.

for Diet. More detailed descriptions and evaluations of self-optimization and self-repair for J2EE applications can be found in [14], and [5], respectively.

5.1 Self-Optimization

The experiment was conducted on a cluster X86-compatible machines connected through a 100 Mbps Ethernet LAN running various versions of the Linux Kernel. The RUBiS application benchmark [2] is used as J2EE application. RUBiS implements an eBay-like auction system and includes a workload generator. Our J2EE software environment relies on Jakarta Tomcat 3.3.2 (Web and servlet servers) [29], MySQL 4.0.17 (database server) [18], C-JDBC 2.0.2 (database load-balancer) [11], PLB 0.3 (application server load-balancer) [22], Sun’s JVM JDK 1.5.0.04 and MySQL Connector/J 3.1.10 JDBC driver (to connect the database load-balancer to the database servers). The machines are distributed as follows: one node for the autonomic management platform, one node for the PLB load-balancer, up to two nodes for replicated Tomcat servers, one node for the C-JDBC load-balancer, up to three nodes for the replicated MySQL servers (backends), one node for the RUBiS workload generator. During execution the number of involved machines varies according to the workload.

To evaluate the effectiveness of the management platform, we designed a scenario which illustrates the dynamic allocation and deallocation of nodes to tackle performance issues related to a changing workload: (a) at the beginning of the experiment, the managed system is submitted to a medium workload (80 clients), then (b) the load increases progressively up to 500 clients (21 new emulated clients every minute) and finally (c) the load decreases symmetrically down to the initial load (80 clients).

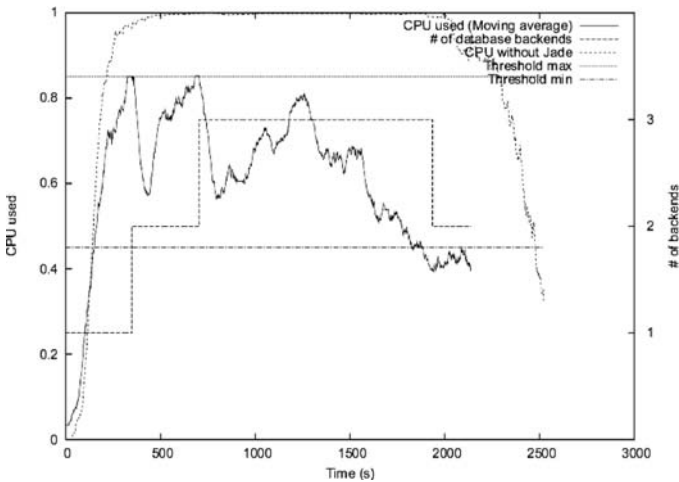


Fig. 9 Self-optimization for the database tier

To quantify the effect of the reconfiguration, the scenario has been experimented without autonomic management, so that the managed system is not resized. Figure 9 presents the result of this experiment on the database tier (although it was experimented at the level of each tier).

All the database backends are monitored by a probe which computes a moving average of the CPU load across all nodes, so as to observe a general load indication of the whole replicated server. The two horizontal lines represent the CPU thresholds used to trigger dynamic reconfiguration (insertion or removal of a database backend). The stair-like curve indicates the number of database backends.

Without autonomic management, the CPU usage rapidly saturates, which results in a trashing of the database. With autonomic management enabled, when the average CPU usage reaches the maximum threshold, the manager triggers the deployment of a new database backend which implies a decrease of the average CPU usage. Symmetrically, when the average CPU usage gets under the minimum threshold, the manager triggers the removal of one backend.

5.2 Self-Repair

The experiment was conducted in a cluster environment similar to that of the previous section. In this experiment we used our platform to deploy a Diet architecture composed of one MA, one LA and two SeDs. The main objective of this experiment is to demonstrate the effectiveness of automatic repair in the case of server failure. Consequently, we artificially induced the crash of a server in the managed system and we observed the load distribution (the CPU usage) on the different servers.

The different machines were distributed as follows: one machine for the MA, one machine for the LA, two machines for the SeDs (computing servers), one machine for the management platform and three machines for submitting client requests (each machine sends 5000 requests). Our experiment uses a DGEMM³ computation of 100×100 matrix.

Figure 10 shows the observed behaviour without using repair management. Both servers (SeD1 and SeD2) have a CPU usage⁴ which stabilizes at approximately 50%, until the crash of SeD2. After the crash, the workload is totally sent to the single remaining server (SeD1), which CPU usage increases rapidly up to 80%.

Figure 11 shows the observed behaviour with repair management enabled. During the interval between the crash and the repair, the CPU usage of server SeD1 increases rapidly since the workload is sent to the single remaining server (SeD1), but only for a short time interval (about 10 s), as the system detects the failure and replaces the failed server by a new server (SeD3). Rapidly, the two servers (SeD1 and SeD3) stabilize at the same CPU usage level as before the crash.

³ DGEMM: a matrix computation which is part of linear algebra problems. It is often used as a point of reference for performance evaluation.

⁴ The reported CPU usage is an average over 5 s.

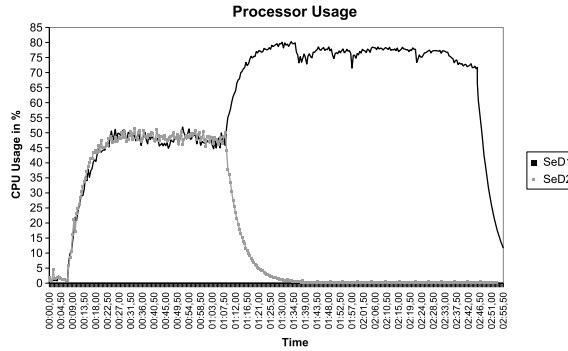


Fig. 10 Failure scenario without autonomic repair

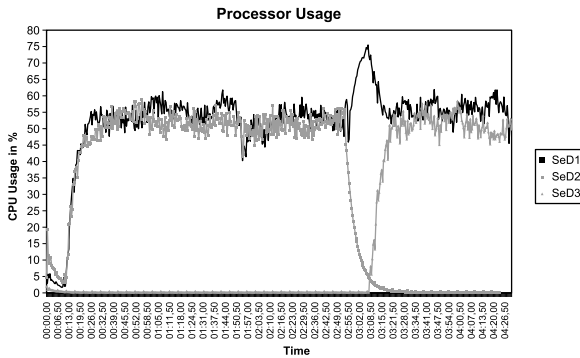


Fig. 11 Failure scenario with autonomic repair

6 Related Work

Autonomic computing is an appealing approach that aims at simplifying the hard task of system management, thus building self-healing, self-tuning and self-configuring systems [17].

Management solutions for legacy systems are usually proposed as ad hoc solutions that are tied to particular legacy system implementations [25, 32]. This unfortunately reduces reusability of management policies and requires these policies to be reimplemented each time a legacy system is taken into account in a particular context. Moreover, the architecture of managed systems is often very complex (e.g. multi-tier architectures), which requires advanced support for its management. Projects such as Jade/Tune or Rainbow [13], with a component-based approach, propose a generic way to manage complex system architectures.

Several projects have addressed the issue of self-optimization and resource management in a cluster of machines. Instead of statically allocating resources to applications managed in the cluster (which would lead to a waste of resources), they aim at providing dynamic resource allocation.

In a first category of projects, the software components required by any application are all installed and accessible on any machine in the cluster. Therefore, allocating additional resources to an application can be implemented at the level of the protocol that routes requests to the machines (Neptune [25] and DDS [33]). Some of them (e.g. Cluster Reserves [4] or Sharc [31]) assume control over the CPU allocation on each machine, in order to provide strong guarantees on resource allocation.

In a second category of projects, the unit of resource allocation is an individual machine (therefore applications are isolated, from a security point of view). A machine may be dynamically allocated to an application by a hosting centre, and the software components of that application must be dynamically deployed on the allocated machine. Projects such as Jade/Tune, Oceano [3], QuID [23], OnCall [19], Cataclysm [30] or [32] fall into this category.

In most of the cases, the autonomic policies have to be programmed using the programming interface of the underlying component model (a framework for implementing wrappers, configuration APIs or deployment ADLs) which is too low level and still error prone. This was initially the case with Jade; we solved this problem with Tune. We proposed a high-level interface which is composed of the following:

- a language for the description of wrappers,
- a UML profile for specifying deployment schemas,
- a UML profile for specifying reconfigurations as state transition charts.

The main benefit of this approach is to provide a higher level interface to the software environment administrator, while considering the management of any legacy software environment. Several projects followed a similar approach, but either as component models' extensions which are still difficult to handle (e.g. Fscript [20]) or for a specific application domain (e.g. parallel computing [1]).

7 Conclusion

Distributed software environments are increasingly complex and difficult to manage, and their administration consumes a lot of human resources. To address this issue, many research projects proposed to implement administration as an autonomic software, and to rely on a component model to benefit from introspection and reconfiguration facilities that are inherent to component models. In this context, we designed and implemented the Jade autonomic administration system and applied it to the autonomic administration of cluster and grid software infrastructures.

Although component-based autonomic systems like Jade have proved to be appropriate, we observed that the interfaces of a component model are too low-level and difficult to use. In order to implement wrappers for legacy software, to describe deployed architectures and to implement reconfiguration programs, the administrator of the environment has to learn (yet) another framework with complex APIs or specific languages.

With Tune, we proposed a higher level interface for describing the encapsulation of software in components, the deployment of a software environment and the reconfiguration policies to be applied autonomically. This management interface is mainly based on UML profiles for the description of deployment schemas and the description of reconfiguration state diagrams. A language for the description of wrapper is also introduced to hide the details of the underlying component model.

This work is still in progress. We are currently extending our prototype Tune to enable various form of reconfigurations (not just invocations on wrapper's methods). Notably, we provide support in state diagrams to allow component re-deployment, i.e. changing a component's location and adding a component instance, while still conforming to the specified abstract deployment schema. We are also metamodelling Tune's policy specification languages in order to formalize their semantics and to generate specialized editors.

Acknowledgments Many of the contributions in this chapter were separately presented in several conference papers [5, 6, 14]. We would like to thank all the contributing authors, especially Fabienne Boyer, Sara Bouchenak, Christophe Taton and Sylvain Sicard. The work reported in this chapter benefited from the support of the French National Research Agency through projects Selfware (ANR-05-RNTL-01803), Scorware (ANR-06-TLOG-017) and Lego (ANR-CICG05-11).

References

1. Aldinucci, M., Danelutto, M., Vanneschi, M.: Autonomic qos in assist grid-aware components. In: 14th Euromicro International Conference on Parallel, Distributed and network-based Processing, Montbéliard-Sochaux, France (2006)
2. Amza, C., Cecchet, E., Chanda, A., Cox, A., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K., Zwaenepoel, W.: Specification and Implementation of Dynamic Web Site Benchmarks. In: IEEE 5th Annual Workshop on Workload Characterization. Austin, TX (2002)
3. Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M.: Oceano - SLA based management of a computing utility. In: 7th IFIP/IEEE International Symposium on Integrated Network Management. Seattle, WA (2001)
4. Aron, M., Druschel, P., Zwaenepoel, W.: Cluster Reserves: a mechanism for resource management in cluster-based network servers. In: International Conference on Measurement and Modeling of Computer Systems. Sant Clara, CA (2000)
5. Bouchenak, S., Boyer, F., Hagimont, D., Krakowiak, S.: Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. In: 24th IEEE Symposium on Reliable Distributed Systems. Orlando, FL (2005)
6. Broto, L., Hagimont, D., Stolf, P., Depalma, N., Temate, S.: Autonomic management policy specification in tune. In: 23rd Annual ACM Symposium on Applied Computing, Fortaleza, Brazil (2008)
7. Bruneton, E., Coupaye, T., Leclercq, M., Quma, V., Stefani, J.B.: The fractal component model and its support in java. In: Software - Practice and Experience, special issue on "Experiences with Auto-adaptive and Reconfigurable Systems", 36(11-12):1257-1284 (2006)
8. Burke, B., Labourey, S.: Clustering With JBoss 3.0 (2002) <http://www.onjava.com/pub/a/onjava/2002/07/10/jboss.html>
9. Buyya, R., Venugopal, S.: A Gentle Introduction to Grid Computing and Technologies. CSI Communications 29 (2005)

10. Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J., Zwaenepoel, W.: Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. In: 4th ACM/I-FIP/USENIX International Middleware Conference. Rio de Janeiro, Brazil (2003)
11. Cecchet, E., Marguerite, J., Zwaenepoel, W.: C-JDBC: Flexible Database Clustering Middleware. In: USENIX Annual Technical Conference, Freenix track. Boston, MA (2004) <http://c-jdbc.objectweb.org/>
12. Combes, P., Lombard, F., Quinson, M., Suter, F.: A scalable approach to network enabled servers. In: In 7th Asian Computing Science Conference (2002)
13. Garlan, D., Cheng, S., Huang, A., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-based self adaptation with reusable infrastructure. In: IEEE Computer, 37(10) (2004)
14. Hagimont, D., Bouchenak, S., Palma, N.D., Taton, C.: Autonomic management of clustered applications. In: IEEE International Conference on Cluster Computing (2006)
15. He, X., Yang, O.: Performance Evaluation of Distributed Web Servers under Commercial Workload. In: Embedded Internet Conference 2000. San Jose, CA (2000)
16. Iyengar, A., MarcNair, E., Nguyen, T.: An Analysis of Web Server Performance. In: IEEE Global Telecommunications Conference. Phoenix, AR (1997)
17. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. In: IEEE Computer Magazine, 36(1) (2003)
18. MySQL: MySQL Web Site. <http://www.mysql.com/>
19. Norris, J., Coleman, K., Fox, A., Candea, G.: OnCall: Defeating Spikes with a Free-Market Application Cluster. In: 1st International Conference on Autonomic Computing (2004)
20. ObjectWeb: FScript <http://fractal.objectweb.org/fscript/>
21. Oriezy, P., Gorlick, M., Taylor, R., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., A.Wolf: An architecture-based approach to self-adaptive software. In: IEEE Intelligent Systems 14(3) (1999)
22. PLB: PLB - A free high-performance load balancer for Unix. <http://plb.sunsite.dk/>
23. Ranjan, S., Rolia, J., Fu, H., Knightly, E.: QoS-Driven Server Migration for Internet Data Centers. In: 10th International Workshop on Quality of Service. Miami Beach, FL (2002)
24. Shachor, G.: Tomcat Documentation. The Apache Jakarta Project <http://jakarta.apache.org/tomcat/tomcat-3.3-doc/>
25. Shen, K., Tang, H., Yang, T., Chu, L.: Integrated resource management for cluster-based internet services. In: 5th USENIX Symposium on Operating System Design and Implementation (2002)
26. Sudarshan, S., Piyush, R.: Link Level Load Balancing and Fault Tolerance in NetWare 6. NetWare Cool Solutions Article (2002). <http://developer.novell.com/research/appnotes/2002/march/03/a020303.pdf>
27. Sun Microsystems: Java 2 Platform Enterprise Edition (J2EE) <http://java.sun.com/j2ee/>
28. Sun Microsystems: Java DataBase Connection (JDBC) <http://java.sun.com/jdbc/>
29. The Apache Software Foundation: Apache Tomcat. <http://tomcat.apache.org/>
30. Urgaonkar, B., Shenoy, P.: Cataclysm: Handling Extreme Overloads in Internet Services. Technical report, Department of Computer Science, University of Massachusetts (2004)
31. Urgaonkar, B., Shenoy, P.: Sharc: Managing CPU and network bandwidth in shared clusters. IEEE Transactions on Parallel and Distributed Systems **15**(1) (2004)
32. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic Provisioning of Multi-Tier Internet Applications. In: 2nd International Conference on Autonomic Computing. Seattle, WA (2005)
33. Zhu, H., Ti, H., Yang, Y.: Demand-driven service differentiation in cluster-based network servers. In: 20th Annual Joint Conference of the IEEE Computer and Communication Societies. Anchorage, AL (2001)

Dynamic WSDL for Supporting Autonomic Computing

Michael Brock and Andrzej Goscinski

Abstract An autonomic computing system is organized into building blocks that can be composed together to form a self-managing system. Architecturally, this matches service-based computing systems, which are the outcomes of the most recent effort to provide interoperability and usability through the use of services. Autonomic computing is an attractive information technology for managers and clients. Rather than overwhelm administrators and programmers with hundreds or even thousands of machines within a distributed computing system, a cloud, autonomic computing systems manage themselves thus freeing some of the underlying tasks from administrators and programmers. When a new component is added, the system learns of it and makes use of the component by itself. When a component fails, attempts are made to automatically recover the component before human assistance is requested. But to support such complexity, an environment rich in current contextual information is needed. Currently, this information has to be extracted via notifications, which themselves require the discovery of a notification management services. For service-based distributed systems, the knowledge of service state and its attributes is of the most crucial importance for the service provision. We propose the application of the innovative Resources Via Web Instances framework, which allows the dynamic state of services to be exposed via their interfaces. The inclusion of state in the interface directly creates an information rich context for autonomic distributed systems without additional complexity of notification mechanics. Furthermore, our innovative framework allows the attributes of a web service to be shown, thus allowing the autonomic system to better cater for the installation and use of new components. With the features of a new component shown as attributes, the autonomic distributed system can take into consideration the nature of the new component.

M. Brock (✉)
Deakin University, Waurn Ponds, Geelong Victoria 3217, Australia
e-mail: mrab@deakin.edu.au

1 Introduction

With the rising size and complexity of today's distributed systems and infrastructures, a faster rising stress is placed on managers, IT administrators and programmers. Thus, the size of a distributed system can be limited by the amount of man power than can be found to support it. As it stands, each node in a distributed system has so much sophisticated software that often each node needs its own administrator to maintain it. Given that the world of IT (at the time of writing) has a severe skill shortage it is hard to find administrators to maintain the systems and programmers who can write good software. This implies that administrators are often given more systems they can handle, thus resulting in high latencies in getting problems fixed and a high error percentage. Furthermore, programmers are forced to get involved and write code that addresses management aspects of computing systems rather than to concentrate on the development of a discipline application. The answer to these complexities lies in autonomic computing [24, 26].

With the rising size of distributed systems, the problems of interoperability and the lack of usability are becoming pronounced. In response to them, the current trend in the development of distributed systems is to use services to govern the access to resources. By encapsulating the resource in a service and accepting all interactions to the resource via the service's interface, clients to the service are freed from needing to know the complexities of the resource. Services are characterized by well-defined interfaces that make a standardized invocation of them possible. This shows that service-oriented architectures (SOAs) of distributed systems match the architecture of IBM's autonomic systems in natural manner.

Autonomic computing is an attractive solution where large distributed systems are able to manage themselves before requiring the attention of administrators, computer technicians or programmers. While this sounds ideal, autonomic computing has requirements of its own; (i) it needs to be aware of itself and be able to learn of new possibilities to improve itself and (ii) it needs to make use of newly found resources and react to resource failures. This means that the autonomic system needs easy access to information about itself, its composing components and other surrounding components. If there is a lack of information or the information is outdated, the autonomic system may spend more time introspecting itself, its composing components and any new components it discovers instead of automatically managing itself. This briefly demonstrates the importance of the Self-Awareness and Self-Discovery characteristics for Self-Management.

The aim of our research is to carry out our study into the enhancements of autonomic distributed systems by allowing components in such systems to expose more information about themselves. A component is any newly provided and/or discovered unit found by the autonomic system. This can vary from a cluster node to a piece of software. In this chapter, the term component is mainly used to describe any unit (software or hardware) that the autonomic system is comprised of or has been recently published and/or discovered, in particular a new service.

The main interest of our research is the exposure of component state and component attributes. State is important to an autonomic system if (for example) it needs

to learn of a failed or failing component. State is also important so an autonomic system can better optimize itself. Attributes are like state, but are better suited to describe the nature of a component and not its activity. For example, state will indicate if a component is working or has failed and attributes will show the nature of the component and possible operation limits.

In our work, attributes describe the nature of a component, such as how it is built and what its limitations are. Attributes are important in autonomic computing as they ease the autonomic systems' ability to discover and install new components. Through attributes, the autonomic component can publish information about itself and the autonomic system can also get a clear view on what the functionality of a newly discovered component is and can even elect to reject it if the component cannot be effectively accommodated. By showing attributes, the autonomic system can quickly see what the component needs and accommodate for them. This is better than current technologies requiring the use of inspection to see the state and attributes of a component.

Our study is in line with the growing trend in other fields of distributed computing, mainly the use of SOA and web services to build large, sparsely distributed and interoperable systems, called clouds. Examples of these are recent computation efforts of Amazon's EC2 [1] and S3 [2] solutions. Virtualization and web services make it possible to access resources such as a database or cluster node through the Internet regardless of the architecture or inner workings of the resource. When we access data on the Internet, we never concern ourselves with how the data are stored or how it is encoded. The same goes with resources where the specifics on how the resource is accessed and managed are hidden by the service thus making the resource easier to use by clients.

The challenge with using web services to build distributed systems is that web services are stateless whereas distributed systems require the knowledge of state to manage resources to provide services to users. The Web Service Resource Framework (WSRF) [16, 35] is an attempt to address the challenge of statelessness of web services. While innovative WSRF did not cover how web services were discovered and relied on the ineffective UDDI (Universal Discovery, Description and Integration) standard [12] for discovery. While interoperable with UDDI, WSRF still left clients with the frustration of discovering services that were no longer available.

We present in this chapter the innovative Resources Via Web Instances (RVWI) [8–11] framework, a web services-based solution that allows services to show their state and attributes through their interface (WSDL, Web Service Description Language) without the need for extensive examination using additional services. We examine and assess how the behaviours of autonomic service-based distributed systems can be improved by creating an information rich context where new components can be added without extensive examination. Furthermore, our solution demonstrates how existing components can easily and rapidly show changes in their own contexts.

This chapter is organized as follows. Section 2 presents related work, in particular autonomic computing characteristics and some projects that tackled autonomic

computing characteristics. Section 3 addresses services and their relationship to autonomic computing components, in general autonomic computing. Section 4 discusses our innovative framework, RVWI, which directly offers through a dynamic WSDL of the service the Self-Awareness and Self-Discovery characteristics in service-based distributed systems. Section 5 presents a summary of the chapter.

2 Related Work

Autonomic computing is seen by IBM [26] as ‘the development of intelligent, open systems capable of running themselves, adapting to varying circumstances in accordance with business policies and objectives, and preparing their resources to most efficiently handle the workloads we put upon them’.

2.1 *Autonomic Computing Characteristics and Architecture*

Autonomic computing systems can be described as having eight principles [26]:

Self-Awareness – The autonomic system, like humans, is aware that it exists, is aware of the components that enables its existence and each component of the system has its own autonomy.

Self-Configuration and Self-Reconfiguration – The autonomic system must be able to automatically assemble and configure itself from startup and must also be able to reconfigure itself when there are changes to its existence.

Self-Optimization – Over time, the autonomic system must be able to evaluate itself and then make decisions as to how it can better optimize itself.

Self-Healing – Should a component fail in an autonomic system, attempts should be made to recover the failed component.

Self-Protection – An autonomic system must be able to perceive a threat and take actions to prevent itself from coming to harm.

Self-Evaluation – An autonomic system is always aware of its environment and is able to make real-time actions in response to changes in its environment.

Self-Discovery – An autonomic system does not keep to itself. It constantly checks for new components and seeks to learn how to integrate them should it find any.

Self-Prediction – When a user makes use of an autonomic system, the autonomic system may be able to predict the needs of the user and transparently allocate resources without the user ever aware of the allocation taking place.

The Autonomic Computing White Paper [24] extends its vision of autonomic computing by moving to large computing systems and states that the difficulty of their system management will become a barrier to their deployment and management. Furthermore, the document addresses directly the architecture of these large computing systems by stating that the right architecture can provide the key to achieving autonomic behavior at the system level and claims that IBM’s approach

builds upon well-established principles of distributed computing and systems management. This allows us to state that our previous research [19, 20] and currently carried out projects [9, 21] are solidly placed on this challenging track.

The White Paper [24] proposes to organize an autonomic computing system into building blocks that can be composed together to form self-management systems. The major aspects of the proposed architecture are interface and behaviour definition, composition and systemwide self-management. To address these aspects and achieve an autonomic computing system, some basic requirements and tasks have been set up.

Firstly, the architecture should not impose requirements on the internal structure of individual components. This implies that the architecture should focus on interfaces and components behaviour. Thus, for example an autonomic system for a business process can directly reflect changing business requirements rather than have the system impose restrictions on the business process. We too, see this is a vital requirement and believe that the interface to autonomic components should also indicate the state and attributes of a component, not just how to communicate with it.

Secondly, self-managing components should aid in simplifying composition. Thus, for example, replaceable components of a business autonomic system should allow supporting of a variety of different business work flows through well defined interfaces. We too seek to simplify composition by removing some of the complexities from service selection. In particular, we seek to hide away services that are too busy or are no longer active. Having those services among other working services adds to confusion (itself a form of complexity) thus making it harder to chose a service.

Thirdly, the management of the whole autonomic system should be made simple. This implies that self-managing components should make composing them into self-management systems easy.

2.2 Autonomic Computing Projects

IBM's Grand Challenge of identifying autonomic computing as a priority research area has brought research carried out for many years on self-regulating computers into focus. We have long identified the lack of user-friendliness as a major obstacle to the widespread use of parallel processing in distributed systems [20]. In 1993 Joseph Barrera discussed a framework for the design of self-tuning systems [5]. While IBM is advocating a 'holistic' approach to the design of computer systems, much of the focus of researchers is upon failure recovery rather than uninterrupted, continuous and adaptable execution. The latter includes execution under varying loads as well as recovery from hardware and software failure.

The projects related to autonomous computing are oriented towards autonomic applications and on systems, in particular distributed systems. An initial survey on some of these projects was conducted and presented in [22].

Anthill (University of Bologna, Italy) [3] is a framework to support the design, implementation and evaluation of peer-to-peer (P2P) applications. Anthill exploits the analogy between Complex Adaptive Systems (CAS) such as biological systems and the decentralized control and large-scale dynamism of P2P systems. An Anthill system consists of a dynamic network of peer nodes; societies of adaptive agents (ants) travel through this network, interacting with nodes and cooperating with other agents in order to solve complex problems. The types of P2P services constructed using Anthill show the properties of resilience, adaptation and self-organization.

Neuromation [33], Edinburgh University's information structuring project, involves the structuring of information based on human memory. The structure used would be suited for organizing information in an autonomic architecture. The structure used is simple, homogeneous and self-referential.

University of Freiburg's Multiagent Systems Project [32] revolves around the self-organized coordination of multiagent systems. This topic has some connections with Grid computing, especially economic coordination issues like in Darwin, Radar or Globus.

The Immunocomputing project [25] (International Solvay Institutes for Physics and Chemistry, Belgium) aims to use the principles of information processing by proteins and immune networks in order to solve complex problems while at the same time being protected from viruses, noise, errors and intrusions.

OceanStore (Berkeley University of California) [36] is a persistent data store which has been designed to provide continuous access to persistent information to an enormous number of users. The infrastructure is made up of untrusted servers; hence, the data are protected using redundancy and cryptography. Any computer can join the infrastructure by subscribing to a OceanStore service provider. Data can be cached anywhere, anytime, thus improving the performance of the system. Information gained and analysed by internal event monitors allow OceanStore to adapt to changes in its environments such as regional outages and denial of service attacks.

The Recovery-Oriented Computing (ROC) [37] project is a joint Berkeley/Stanford research project that is investigating novel techniques for building highly dependable Internet services. ROC focuses on the recovery of the system from failures rather than their avoidance.

A Grid scheduling system, developed at Monash University, called Nimrod-G [34], has been built to provide tools and services for solving coarse-grain task farming. The resource broker/Grid scheduler has the ability to lease resources at runtime depending on their capability, cost and availability.

The Bio-inspired Approaches to Autonomous Configuration of Distributed Systems [6] at University College London have used bio-inspired approaches to autonomous configuration of distributed systems (including a bacteria inspired approach).

Although their study show that some form of data collection is in place (hence leading towards Self-Awareness and Self-Discovery) these two characteristics are not addressed explicitly, and another work [38] has even gone as far to say that Self-Awareness is a minor characteristic. In human anatomy, suddenly finding a foreign object (both material and biological) is not minor. If it is not minor in biology, why

should it be minor in autonomic computing which seeks to bring the self management features of biology to computing?

One work of interest is how event management is addressed in [40]. To effectively carryout Self-Awareness, the autonomic system needs to carefully consider what events have happened and what appropriate action to take. While a need is stated to effectively match an event to an optimal action, it is not stated how the event is captured and examined in an autonomic system. The work presented a case study where autonomy was placed in telephony network.

Unity [13], while not a fully Self-Aware, autonomic system, is a step in the right direction to making autonomic systems aware of themselves and their various components. In Unity, each individual component has its own autonomy: thus it is able to be placed in an environment, learn the context, find resources, configure itself and finally begin execution and start offering its services.

The term used in Unity is Self-Assembly; as the name states, when an autonomic component starts, it works with the existing autonomic system to assemble its needed environment together and is even able to see if it is running out of resources and request more resources from the autonomic system. Unfortunately, while each component is autonomic the autonomy does not carry on to a higher level. An autonomic component is needed to store policies which are placed there by an administrator. Furthermore, while each component has an interface, the state is only visible via communication with the autonomic component and communication is only allowed with the component if it has authorized a relationship with another component.

While many of these systems engage in some aspects of Autonomic Computing, none engage in research to develop a system which has all eight of the characteristics required. In response to this problem, a technology on building autonomic cluster operating systems was developed and an autonomic cluster operating system as a proof of concept of this technology was built [22]. This system has been built from scratch to offer an autonomic non-dedicated cluster by providing availability, parallelism management, transparent fault tolerance and easy programming. This system relieves developers from programming operating system-oriented activities, and provides to developers of next generation application software both message passing and distributed shared memory.

While conducting a study in literature we found that there is much activity in autonomic computing and that most of the activity is centred on Self-Healing, Self-Optimization and Self-(Re)Configuration. We are strongly convinced that not enough attention is paid to the Self-Awareness and Self-Discovery characteristics as they provide information to make decisions.

3 Services vs. Autonomic Computing

The lack of research attention to Self-Awareness and Self-Discovery is a concern as without them there can be no awareness hence no information to detect fault, sudden changes in system state or even the addition of a new component. Also, due to the

lack of contextual information, Self-Configuration and Self-Optimization is made difficult as neither characteristic can find the information needed to effectively carry out their roles in managing the autonomic system. This means that Self-Healing, Self-Optimization and other principles have to have their own detection mechanics built in. While the principles are supported there is no global view on the state of the autonomic system nor is there a global view of the environment of which the autonomic system exists. The question is how the lack of attention to these two characteristics of autonomic computing could be addressed in distributed systems of today.

For a distributed system to function correctly, application components (often encapsulated as a programming object) executing on different computers throughout a network must be able to communicate. The current trend in the development of distributed systems is to use services, which expose resources behind them, as building blocks for the distributed system. An early example of this trend can be seen in the Grids [17, 42] while a very recent example can be seen in EC2 [1]. Services are characterized by well-defined interfaces and the invocation of them made possible by the defined interfaces. This shows to us that SOAs of distributed systems match the architecture of IBM's autonomic systems in natural manner: an autonomic component in an autonomic system can be considered and interpreted as a service in a service-based distributed system.

An analysis of the current development and trends in the area of IT and computing demonstrates that a stronger emphasis should be put on autonomic computing based on distributed systems and linked to service computing.

3.1 Self-Discovery and Self-Awareness Issues

Of interest to this chapter are the principles of Self-Discovery and Self-Awareness. Self-Awareness allows a system, object or a service to learn about its state and attributes. This information is necessary to provide a correct decision on time. For example, current and easily accessible state information is needed to promptly detect a failing component in the autonomic system. The knowledge of state in a distributed system, or any computing system, is necessary for its management. A distributed system, as stated earlier, is composed of a set of components, which cooperate closely using messages to achieve the goal of the whole systems. Thus, the knowledge of cooperating components is of the most crucial importance.

Self-Discovery is where the autonomic system is able to learn about the environment it is working in, of new components that have been added and components that may have failed or have been removed. Even with the use of open technologies, such as web services, all that can be learned is the interface, thus it can only be learned how to use a web service. The same issue exists with autonomic systems when they find a newly discovered component and try to make use of it.

The frustration with autonomic systems is that only the knowledge of the component operations is readily available (as well as what data each operation require and

what data are returned). What cannot be learned is the required execution platform of the component or the intention of the component. Without this information the autonomic system cannot safely make use of the component. The autonomic system has to lose valuable time communicating with the component to learn what it needs in terms of execution environment, maintenance facilities, etc.

Self-Awareness, where the autonomic system is aware of its own state and the state of its comprising components, is a basic requirement. When an autonomic system discovers a new component, the component needs to be able to send routine notifications about any changes to its state. An autonomic system cannot routinely request the state of any of its components. The reason for this is the rise in the number of components that will lead the autonomic system to spending more time examining components than managing itself.

Autonomic computing is the vision of making complex computer systems easier to manage. Unfortunately, autonomic computing itself is complex and care must be taken otherwise more complexity gets layered on an already challenging complexity. As stated in at the start of Sect. 3, the current trend in the development of distributed systems is to use services and compose the whole distributed systems using services. We look to SOA and web services to take some of the complexity out of autonomic computing for us.

3.2 The Service-Oriented Architecture

For a distributed system to function correctly, application components (often encapsulated as a programming object) executing on different computers throughout a network must be able to communicate. However, heterogeneity makes this communication difficult. In the early 1990s, many companies and organizations realized the need for such functionality and began developing their own technologies to enable communication among distributed components, e.g., OMG's Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) and Sun Microsystems's Remote Method Invocation (RMI) [41]. Unfortunately, interoperability (the ability to communicate and share data with software from different vendors and platforms) is limited among these technologies. A solution to the problem of interoperability is SOA [27].

SOA is an approach to designing and implementing a large distributed system. Instead of building a system as a large monolith, the system is built as a collection of inter communicating services. SOA is an architectural style that supports service orientation, which is a way of integrating a business or a set of businesses as linked services and the outcomes they bring. Services are created, made available and consumed by users (clients). Services are independent, stand-alone, generalized and re-usable and exist somewhere accessible, for example, to check a customer's credit, or open a new account. In general, a service can be anything from a high-level 'business process' to one of very many low-level functionalities, shared among other processes.

The idea of SOA is to encapsulate each resource (database, human knowledge, computer node, business task, etc.) in the system with its own service and to have all interactions with that resource via the service. It is also possible to chain services together to form a work flow where data go from one service to another, undergoing a small amount of processing on each service.

The advantage of SOA is that services can be added and removed depending on the overall needs of the distributed system or the needs of the organization maintaining it. Services also improve usability where the management and operational aspects of a resource are hidden by the service. This means the client can make use of a resource immediately and without having to go through other management processes.

The main problem of SOA-based (and even distributed) systems is: how does the client (be it another service or remote user) find a needed service? SOA-based systems (as well as distributed systems) need discovery services to help locate needed services. Figure 1 shows the typical structure of discovery in SOA-based systems.

The first step is the publication of a service to a discovery service. This can be done either by the service itself or by the Service Provider (the owner and maintainer of the service). Next, the client contacts the discovery service for information on a service its needs to perform a task. In this step, the client will ask, 'I need a service that can multiply two matrices, each two thousand elements in size'. The Discovery Service returns the location and interface of any services it finds to the client. The client then uses operations listed in the interface to learn the state and attributes of the service by approaching the web service before using it.

This implies that if the state of the first service approached indicated that the service is busy, the client tries accessing the second, third and so forth, learning the state and attributes of each service along the way. Checking the state and attributes of services takes a lot of time and wastes communication bandwidth.

Another major frustration with SOA in the past was defining and implementing how the services in SOA systems communicated with each other and with clients outside the SOA-based system itself. As SOA left communication semantics to SOA developers, it meant that each SOA-based system had its own means of communication thus making it harder for clients to communicate with multiple SOA-based

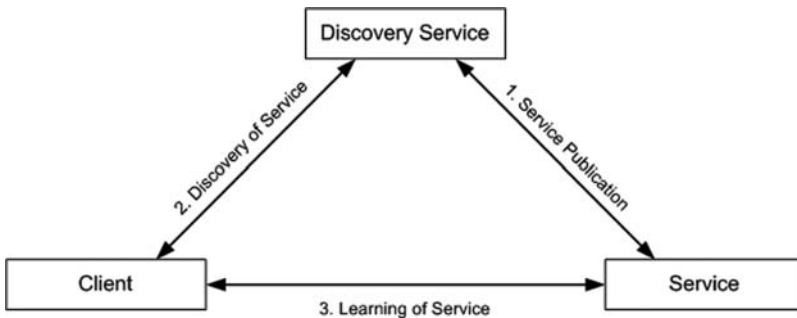


Fig. 1 Typical model for SOA discovery

systems. The client needed to contact the developers of the SOA-based systems to learn how to communicate with the system. This means that the problem of interoperability has not been completely solved.

3.3 *Web Services*

Recently, web services have been used to implement SOA-based systems thanks to their interoperability features. An early example can be seen with OSGI [42] and more recently in version 4 of the Globus Toolkit [17]. Unlike DCOM and CORBA, web services operate using open (i.e., non-proprietary) standards. It is possible to say that Web services technology represents the next stage in distributed computing. Thanks to using a service to hide away the architecture and management of a resource from clients, clients can spend more time using a resource instead of having to learn how to use it and how to comply with management policies.

Web services encompass a set of related standards that can enable two computer applications to communicate and exchange data via a network, such as the Internet. Web services are also effective as they make use of open standards and protocols: XML [7], WSDL [14], SOAP [31] and UDDI [12].

XML (eXtensible Markup Language) is a specification on how to describe data using text and in a platform-independent manner. If there is a data structure to share with other software solutions, instead of handing developers a language-specific library to use the structure, one can describe the structure in XML and from it developers can reconstruct the data structure in their choice of language. In web services, XML is used to describe the data and the structure of the messages to go to and come from the web service.

The WSDL is a XML-based language for creating WSDL documents that describe the web service. The WSDL describes the messages needed to communicate with the web service, the data that goes into the web service and the data that comes out of the web service. In short, the WSDL describes the functionality of the web service but as we explain when we introduce RVWI, it does not describe the state and attributes of the web service. Being XML-based, developers can build clients to a web service simply by reading the WSDL. This means that a web service written in C++ can have a client to it build in Java simply by using the WSDL and nothing else.

SOAP is a standard for defining XML-based messages on how a software component on one machine can communicate with a software component on another machine without having to know how each software component is executed on either machine.

Finally, UDDI, a standard on how to make a web service that finds other web services for clients. A UDDI service keeps the location, WSDL and one or more bindings for which clients are built to make use of the web service. The approach of UDDI is like a phonebook where service providers publish information about their services and clients can then easily access it.

Web services take advantage of object-oriented programming techniques in that web services enable developers to build applications from existing software components. The technology encourages a modular approach to programming. Thus the Internet is transformed into an enormous library of programmatic components available to developers; some developers can use available components, whereas other developers may make their services available. As a follow-up, web services can enable any two software components to communicate regardless of the technologies used to create each component or the platform on which the component resides.

3.4 SOA and Autonomic Computing

SOA is a powerful architecture in large distributed system design. The problem is how to implement the SOA-based architecture. The response is web services: a collection of open standards and powerful technologies to make large distributed systems from various components created from various methodologies. The two together create flexible, reusable, scalable and reliable distributed systems. Unfortunately, neither SOA nor web services offer a means of managing the systems made using them. It falls on the developer of the distributed system to implement infrastructure to manage the distributed systems themselves.

One solution that combines SOA, web services and Autonomic Computing principles is the Service Management Broker (SMB) [29, 30]. Of interest to autonomic computing are SMB's capabilities for Self-Healing and Self-Awareness. Self-Awareness is vital to Self-Healing: if the autonomic system is not aware of itself, it cannot be aware of any faults within itself either.

Of interest to this chapter is how SMB addresses Self-Healing. SMB uses error and fault detection akin to a heart rate monitor. In medicine, a heart rate monitor routinely detects the pulse of a patient's heart and signals an alarm if the heart stops. In SMB, all services are required to routinely send a signal to a designated monitoring service. If a service fails to send a signal, the fault detection services activate to learn what has become of the service. If the service has failed the Self-Recovery systems restore the service. All this happens without the user of the server ever knowing that the service has failed.

SMB is a pure web services solution. Often, web services are not used in large distributed systems because the technology itself is still very young and standards only recently started to become finalized in the past couple of years. SMB is proof that web services are a very mature platform for building large distributed systems as well as autonomic systems.

This shows that there is architecture, technology and its proof of concept that allows the provision of autonomic features to service-based distributed systems. However, a weakness of this unique approach is that state and attributes of a web service are not easily and quickly accessible to clients (users and/or other web

services). This means that Self-Awareness and Self-Discovery in service-based distributed systems require additional work.

3.5 *Towards Stateful Web Services*

While very interoperable, web services have two main issues: they are stateless and UDDI discovery is inaccurate. SOA-based systems (or any form of distributed system) need services to have state so that requests that depend on the outcomes of previous requests are possible.

The state of services is just as important as the interface itself. When learning of services, clients should know about the state of services at the same time as they do about the interface. Clients should not have to learn that the service is too busy or unavailable after spending time learning about the interface and what methods could be used to best represent the state. UDDI is also inaccurate as it assumes that the information given to it never changes over time and UDDI does not consider the state of a web service when clients consult it.

A first attempt at making web services stateful was the Open Grid Service Infrastructure (OGSI) [42]. OGSI acted as a layer on top of web services [28] and indicated how the state of a web service was created, used, maintained and destroyed. OGSI specified that (i) Web services and their state were kept separate from each other; (ii) State was kept in a uniquely identifiable unit called an instance; (iii) Instances are identified by an Endpoint Reference (EPR) and (iv) The EPR is used to load the instance from persistent storage before the web service performed operations for the client.

When clients first use an OGSI service an Instance is created for them and the EPR returned. When the clients make later request, the EPR is supplied so the correct instance can be loaded into the web service.

While OGSI preserved state, the state itself was not directly accessible from the service. OGSI required a designated service, such as the Monitoring and Discovery Service (MDS) [18]. MDS kept track of all services through notifications. Whenever the state of a service changed, MDS was notified thus keeping itself current. While the state was accessible in the MDS, if MDS failed, the state could not be learned without using methods in the service. Furthermore, MDS is too complex to be used by non-computing professional.

An attempt was made in [39] to make state more easily accessible using the UDDI Service [12]. UDDI was itself a web service thus any form of client could make use of it. UDDI was expanded to track state in the same manner as MDS thus information on state was kept current. The enhanced UDDI service was able to receive notifications from service providers and their web services thus keeping UDDI current. Unfortunately, it required additional effort from the service providers and clients and did not provide the notification mechanics to the services.

Also, the single point of failure in systems based on OGSI still existed in this solution and again, it was difficult for non-computing experts to make use of it.

The Web Service Resource Framework (WSRF) [16] was a step forward for web services. It is a standard on describing how to make web services stateful via an open protocol. The standards brought together by WSRF are WS-Resource [35], WS-BaseNotifications [23], WS-Topics [43] and WS-Addressing [4]. The invention of these notification standards and WSRF made it possible to build web services that was stateful and was easily accessible by clients.

Functionality wise, WSRF was the same as OGSF. The difference was WSRF used web service standards while OGSF added specific functions on top of web services. Also, WSRF partitioned stateful functionality over multiple web service standards [15], while OGSF was a monolithic standard. Like OGSF, WSRF uses a unit called an instance which holds the state of a web service and is kept in persistent storage when not in use. When clients use WSRF services for the first time, an instance is created. The instance also has a unique identifier which is given to the client. That way, the client can inform a service of the state needed and the service can then easily retrieve it.

Again, instances also had unique identifiers called EPRs which are assigned when the instances are created. When the client makes later requests the client supplies the EPR so the web service can load up the correct instance. EPRs are used as clients cannot be easily identified due to their diversity and the nature of some computer networks. Thus the EPR is used both to load an instance and to relate a client to that Instance. The only disadvantage of WSRF was that state still required an additional service to be accessed.

While innovative WSRF did not cover how web services were discovered and relied on the ineffective UDDI standard for discovery. WSRF still left clients with the frustration of discovering services that were no longer available. Also, while the state was preserved, it was kept private to the client the state related to. Furthermore, to state-enable discovery services, such as UDDI, the services had to be heavily modified and the web services retrofitted to send updates on state to the discovery service. Furthermore, if the discovery service failed access to the state of a web service was also lost as the state was only accessible through UDDI and not the service itself.

An innovative framework, furthering the capabilities of WSRF was recently created. The Resources Via Web Instances (RVWI) framework [8–11] is a pure web services framework that adds capabilities on top of WSRF. The main feature is RVWI granting to web services the ability to show their state via their WSDL documents. The innovation of the RVWI framework is that it directly links state and attributes of a web service, in particular a resource behind it that can change state between requests, with WSDL that is the most frequently used object of web services. The significance of this framework is that it improves resource discovery; (i) clients can learn about resource state and attributes when they receive information about the location and the interface of a web service; (ii) the state is always accessible even when discovery services failed and (iii) time and communication bandwidth is saved because there is no need for accessing web services to learn about their states and attributes.

4 The Resources Via Web Instances Framework for Autonomic Computing

We demonstrate in this section the RVWI framework as it allows web services to publish and make directly accessible their state and attributes. More precisely, the state and attributes shown are the state and attributes of the resources behind the web service. Information about changes is conveyed when they occur: thus, they are current and support making informed decisions. The RVWI framework supports the provision of the Self-Awareness and Self-Discovery characteristics in service-based distributed systems.

4.1 SOA Enhancement

When it came to making use of a web service, a lot of time is wasted learning the state and attributes of the web service only to find that it is either too busy or it is too slow in getting requests processed. What is needed is a way to keep the state and attributes of a web service easily accessible and that the current state is always published.

We decided to provide the state and attribute information in the WSDL and keep it current so clients were never left short handed. We made use of the WSDL document because it is the most commonly accessible object of a web service. The WSDL itself is often used by other services such as UDDI so they can inform clients how to use the service. By enhancing the WSDL of the service, any state information quickly propagates to other services and subsequent clients. The bundling of both the state and the WSDL in the one document means clients can quickly see how to use the given service and if it is ready for their request or not. Figure 2 illustrates what we proposed when we created RVWI.

Like with the traditional model for SOA (shown earlier in Fig. 1), services are first published with a discovery service (DS) with its current state (1). Additionally, the Service is able to send updates about its state and attributes to the DS (1a). The

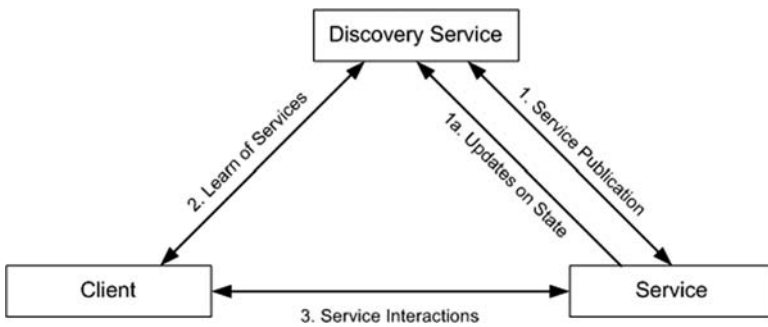


Fig. 2 Updated model for SOA discovery

service represents the resource thus the state of the resource also has to be represented via the service. When the Client contacts the DS (2) the Client learns everything about the Service: service location, message pattern, interface, state, attributes, last down time, etc. Finally, the Client makes use of the Service immediately (3).

The changes RVWI makes to the WSDL are significant. Like the typical SOA model, services are published to a discovery service; but RVWI allows for web services to send updates about themselves to discovery services long after the service has been published. Thus both the web service's WSDL and the discovery service itself remain current to the state of the resource exposed via the web service.

When the client contacts the discovery service, the client is able to get the active state and current attributes of the web service as well as its interface and location. Furthermore, the discovery service can employ one additional step and ensure that web services under a heavy load are not returned to the client.

With the information on current service state and current service attributes, the client can accurately pick a service (if more than one is listed) and then use the service immediately and does not have to waste time learning the state and attributes of the service. Aside from saving time, communication overhead (the time lost from the creation, transmission and processing of messages to and from web services) is also greatly reduced.

The introduction to the RVWI framework shows that it is possible to efficiently provide Self-Awareness and Self-Discovery of services that form a distributed systems.

4.2 WSDL Modification

The modification of the WSDL is not done via an additional service; the WSDL is modified to show the state and attributes from its own service. Figure 3 shows the steps taken to modify the WSDL. As per WSRF, the state of a web service is kept in persistent storage. According to the RVWI framework, the client requests the WSDL (1) the web service generates the WSDL naturally (2). The next step for the service is to find any instances of itself from persistent storage (usually this is a database but can be a file system or any form of storage depending on the implementation) (3). If any instances of the web service are found, the WSDL is modified to include them (4). Finally, the WSDL is returned to the client like any other WSDL document.

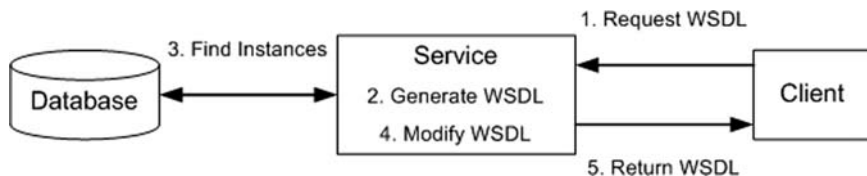


Fig. 3 WSDL modification

```

<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  <rvwi:Instances xmlns:rvwi="RVWI Namespace URL">
    <rvwi:InstanceInfo EPR="Instance EPR String">
      <rvwi:State>
        <rvwi:Description Name="ElementName">
          <rvwi:Value type="ElementType">ElementValue</Value>
        </rvwi:Description>
        ...Other Possible State Elements...
      </rvwi:State>
      <rvwi:Characteristics>
        <rvwi:Description Name="CharacteristicName">
          <rvwi:Value type="CharacteristicType">CharacteristicValue</Value>
        </rvwi:Description>
        ...Other Possible Characteristics...
      </rvwi:Characteristics>
    </rvwi:InstanceInfo>
    ...Other Possible Instances...
  </rvwi:Instances>
  <wsdl:types>...</wsdl:types>
  <wsdl:message name="MethodSoapIn">...</wsdl:message>
  <wsdl:message name="MethodSoapOut">...</wsdl:message>
  <wsdl:portType name="CounterServiceSoap">...</wsdl:portType>
  <wsdl:binding name="CounterServiceSoap" type="tns:CounterServiceSoap">...
    </wsdl:binding>
  <wsdl:service name="CounterService">...</wsdl:service>
</wsdl:definitions>

```

Fig. 4 New WSDL with state information

To include the state and attribute information, the WSDL itself has to be expanded. As the WSDL of a web service is written in XML and is a modular document, it is possible to include additional information about a web service by simply adding a new XML section. Figure 4 shows an example WSDL with the new instances information element.¹ Additionally, RVWI shows the attributes of a web service. Attributes describe the nature of a web service. For example, a web service whose resource is a file will have an attribute on the maximum possible size the file can be.

The WSDL document of a web service has multiple sections, each section describing a small aspect of the web service. For example, all information on the structure of the messages used to communicate with the web service is in its own section. The same goes for the required data structures and the operations web service offers to allow access to the web service. Thanks to the use of sections, it is possible to add additional information to the WSDL by encapsulating it in its own section.

To show the state of the web service in the WSDL, RVWI created a new section in the WSDL called the Instances section. Like how there is a section for each message and operation of a web service, there is now a section for all instances of the web service. The Instances section itself has one to many InstanceInfo sections. This is because stateful web services can have multiple instances. In WSRF, an instance of a web service can have one or many resources; for RVWI we keep them specific where each instance exposes a given resource. Figure 5 shows an example

¹ Note: The RVWI specifications mention state and characteristics instead of state an attributes. To prevent conflict with autonomic characteristics, RVWI characteristics are referred to as attributes.

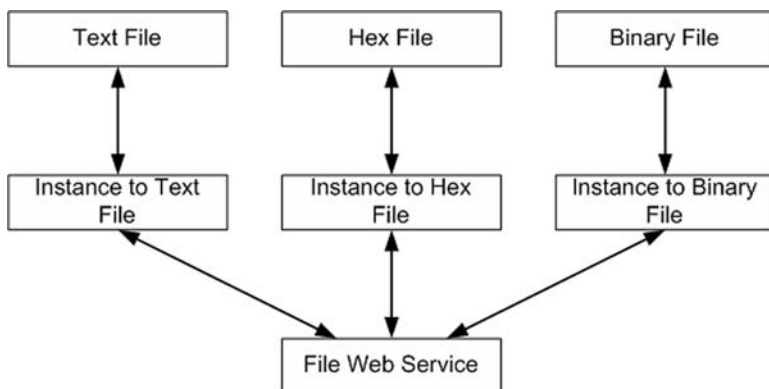


Fig. 5 File resources shown through instances

of multiple resources made accessible through the single web service thanks to the Instances.

In Fig. 5, we have three types of files, Text, Hex and Binary. Each file will have a maximum size, current file size and even state information on what data are being read or written to them. The instances hold this information on behalf of the web service. As stated before web services are stateless and are made stateful through decoupled instances.

All three files are made accessible through the single file web service. If the client wishes to use the Hex File, the instance to the Hex File is loaded thus making the web service (to the client) aware of the current state of the Hex File. In RVWI, instances are used like snapshots of resources, which the web service shows via the WSDL.

The Instances section that RVWI introduces to the WSDL consists of one to many InstanceInfo section. An InstanceInfo section has an ERP attribute and two child sections, the State section and the Attributes section.² As each InstanceInfo relates to an Instance, the EPR is given so the instance can be selected. As the name implies, the State section shows the state of the web service (based on a given instance). Elements such as the current activity of a node on a cluster are shown in this section. The Attributes section describes the nature of the resource. Elements such as the CPU of a node in a cluster are described in this section.

To publish the state and attributes, the State and Attributes sections both have one to many Description elements. In RVWI, it was seen that state was very complex and could not be described in just one element. The state of a cluster could not be simply described as being heavy or light, nor could it be described as being X per cent used. The variations in each node in the cluster and the number of nodes working and have failed all contribute significantly to the state of the cluster. Thus the state in RVWI is described via a collection of elements, all making up the whole state.

² This diagram was taken from the original RVWI literature and updated to have Attributes instead of Characteristics.

While the Attributes sections has one to many Description element, they contrast the State section as each Description element in the Attributes section is its own unit and is not part of a whole. For example, the type of CPU of a node in a cluster is not dependant on any other node.

The use of Description elements in RVWI makes it very fine grained and flexible in describing the resources behind web services. Before, when one wanted to use a cluster, one could only ask, 'I need a cluster of 200 Windows Nodes'. While the client would get a cluster, half the nodes may have been too busy to be used. Thanks to RVWI, the user can see immediately what nodes are free and what nodes are busy. Based on this informtion, a node selection of activity could be made autonomic easily.

4.3 Supporting Dynamic State

As well as showing the state and attributes, the exposed state and attributes also have to be kept current to the resource behind the web service. In WSRF, when the stateful web service completes an operation any changes in state and attributes are placed in the instance and the instance is then placed in persistent storage for later use.

The question raised then is: if RVWI is using an instance to show a resource, how does RVWI ensure that the instance remains current to the resource? This is an important question as the resource is separate from the instance, thus can be subjected to change which will go unnoticed by the instance. The inconsistency between the resource and the instance can cascade to the web service where it will expose inconsistent state and attributes of the resource.

The relationship of clients to web services and resources is shown in Fig. 6. Like with any web service, the Client makes requests to the Web Service (1). This request will likely have an EPR to an Instance which the Service will load at the start of the request and save back when finished (2). Behind the Service is a Resource which the Service exposes. After accepting a request from the Client, the Service performs operations on the Resource (3).

The steps taken in Fig. 6 only supports resources that do not change their state and attributes between client requests. As the Instance keeps some 'cached' informa-

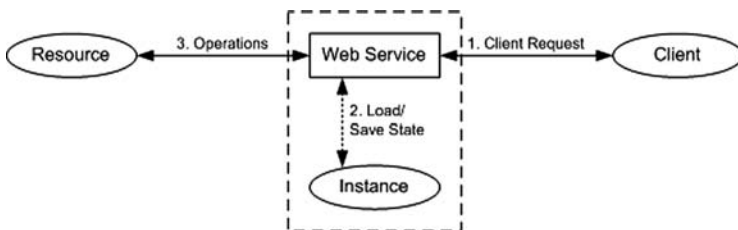


Fig. 6 The Client in relation to Web Service and Resource

tion about the resource, the Instance becomes unusable if the state of the Resource changes.

To keep the web service informed of any changes, a Connector is proposed to track any changes in state and attributes with the Resource. The Connector is simply a piece of software that detects changes in a resource and then communicates that change to the web service exposing the resource (as well as its state and attributes). Changes in state happen more often than attributes. This is because attributes describe the nature of resources, which seldom if ever change. Figure 7 shows the Connector in relation to the Resource and the Service.

When the Resource undergoes a change in state or attributes, it is detected by the Connector (1). The Connector informs the web service about the change (2). The change is reflected in the Instance by the Service (3). Later, when a client asks for the WSDL of the Service, the Service returns a stateful WSDL with all state and attribute information contained in it. The client can then decide whether or not to make use of the web service based on the value of the state and attribute information. The client does not incur the communication overhead introspecting the service for state and attribute information.

The Connector detects changes in a resource via two ways: by letting the resource send signals to it directly or by routinely examining the state of the resource for changes. The reason for communicating the change of the resource to the web service and not the instance is the instance maybe in use when a change in the resource occurs. If we changed the instance while it is in use, and the web service then completes, it will write over and changes we make thus resulting in a lost update. Also, using the web service means that we have a central point where all changes go through, thus serializing updates.

By keeping the state information current, the WSDL itself also remains current to the state of the service. The Connectors track the state of resources and keep web services current with the resource (thus showing correct state in the WSDL) and notifications (provided by WS-N) are used to keep other services up to date with changes in the web service.

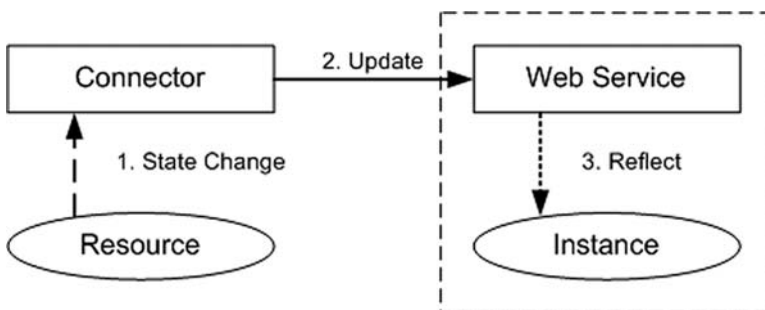


Fig. 7 Using the Resource Connector

4.4 RVWI and WSRF

The RVWI framework was created to allow the state (and attributes) of a web service to be shown in its WSDL and to allow for any service or client holding a copy of the WSDL to receive updates on state to keep their copies current. This addressed the problem often encountered when clients sought web services through discovery services only to find that the service was unavailable and that they had now lost valuable time.

The WSRF standard was used as a foundation for RVWI. In using WSRF we did not have to re-invent a stateful web service framework and by building on top of it, we make our own solutions compatible with other WSRF compliant solutions. Figure 8 shows where RVWI sits with the stateful web service and other web service layers.

WSRF provided all the essentials: state management, addressing and notifications just to name a few. As the functionality was already built and tested, it made sense to further the use of WSRF to show the state of a web service. The challenge at the time was how to make use of the instance. While the state was held, we also wanted to keep attributes. WSRF only considered state thus any additional information could not be told from the state.

To resolve this, RVWI makes use of the facilities in WSRF to help keep state and attributes separate. RVWI simply adds ‘tags’ to the attributes so that when stored in the instance, the attributes are grouped as either state or attributes. Thus it is possible to look at the instance itself and see what makes up the state of the service and what makes up its attributes.

It should be noted that while we are working with stateful web service technologies, we still consider the stateless web service technologies (the bottommost layer). The reason for keeping access to the lower layers, as can be seen again with WSRF

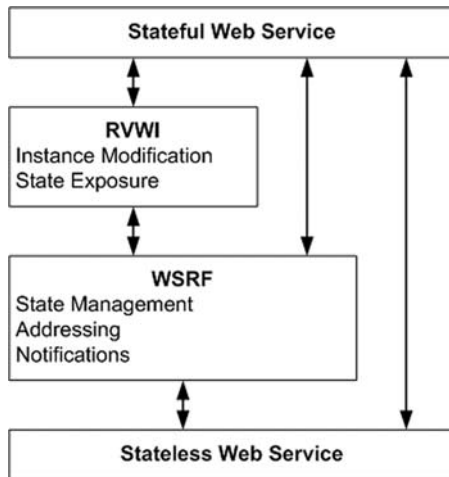


Fig. 8 Stateful web services layers

and RVWI, is to keep interoperability with previous web services. Had we gone the fully layered approach (where each layer was completely covered by the next layer up) we deny ourselves interoperability with stateless web services.

4.5 Supporting Self-Awareness and Self-Discovery via Dynamic WSDL

With the exposure of state and attributes through the WSDL, RVWI is a strong candidate for implementing service-based autonomic distributed computing systems; where resources are made easily accessible via services, particularly web services. The principle of Self-Discovery requires the autonomic system being able to find components (or services) on its own, learning how they are to be used, interoperating them with other components and then effectively making use of them. If an autonomic system discovers a new component, the lack of state and attribute information means the autonomic system spends more time learning about the component before making use of it. If the autonomic system cannot learn the nature of the component or service, how can it make use of it? It would be like trying to incorporate a cluster node without being able to configure it.

State is a vital factor in this principle as the autonomic distributed system should not make attempts to interoperate a component that is in a failed state. As there is a high overhead in facilitating, installing, configuring and activating a component, that overhead becomes wasted if the component is later discovered to be inoperative. Once found to be inoperative, the principle of Self-Healing (where the component is removed and the system reconfigured again) and Self-Optimization triggers thus resulting in even more lost overhead.

Self-Healing and Self-Optimization prove that the principle of Self-Awareness is the most important principle of all. Just like how we humans come to know ourselves to better improve ourselves, autonomic systems need to do the same to keep themselves optimised and working efficiently. This requires an information-rich context where data about the autonomic system and its composing components are kept current and in depth. RVWI (in particular the dynamic WSDL) supports this through the exposure of state and attributes. As the exposure is on the service itself, the exposure is 'always on' and does not require a specialized mechanic to extract.

In autonomic computing, RVWI can support Self-Awareness where the state and attributes of the component are visible. While it is preferred to have a centralized approach (where state and attributes are placed in a indexable service), the current state and attributes should be visible all the time. Thus if an autonomic component is having trouble showing its state, another component can see this and possibly help by pointing the former component to another service.

When it comes to adding a new component to a distributed system, humans ask these questions: What does this component do? What does it need? How is it used?

Currently, only the last question is addressed, if at all. The question of how a component is used covers the operations a component offers and the input and output data for each operation. When making use of a new component, the autonomic system needs to know:

- What the component does, otherwise, it will continue to add one component after another and only use a select few.
- What it needs. If the component is a piece of software, the autonomic system needs to know its execution environment and required software (such as drivers) so it can be used. If the component is a cluster node, the autonomic system needs to know the hardware architecture so it can prepare the node with additional software so the node can quickly accept incoming cluster jobs.
- How the autonomic system is used so only proper data go in and come out of the component.

It is RVWI's ability to show attributes that best address the first two questions. When describing attributes through the dynamic and stateful WSDL, attributes such as the architecture of a cluster node, the maximum size of a data file and even the number of available nodes in a cluster are shown in a form that is easily available to other clients and services.

The attribute exposure in RVWI was originally implemented as a means to further describe the state of a resource exposed via a web service. For example, the dynamic WSDL could show the state of a cluster as having half of its nodes busy. The attributes can further enrich the state exposure by indicating from each free node its architecture, the number of cores in the CPU, its maximum memory and maximum storage, etc.

RVWI (thanks to its creation of dynamic WSDL) lends itself to autonomic computing as attributes of a newly discovered component, such as what it does and what it needs, are now easily exposed. With this additional information, the autonomic system can easily and effectively accommodate the deployment of the newly discovered components. Furthermore, the exposure of state with no need for a centralized manager means the state of all comprising components is easily accessible.

It is natural to state in summary that the RVWI Framework, in particular the dynamic WSDLs, supports the Self-Awareness and Self-Discovery characteristics in service-based distributed computer systems, particularly clouds.

5 Summary

As the size of distributed systems rise, so does the sophistication and complexity of the processes that manage the distributed computing system. With the introduction and use of service based distributed computing systems the management problems of these systems have become even more important. Finding a solution to these problems is even more urgent these days. Autonomic computing is an attractive

technology for managers, system administrators and programmers as it allows large distributed systems to manage themselves with little or minimal human intervention.

But such automation towards self-management has requirements of its own: mainly the need for keeping information about itself, its comprising components and any newly discovered components current and easily accessible. Without easy access to information about the attributes of autonomic components, autonomic systems and clients could lose valuable time incorporating newly discovered components which cannot be effectively used and can even lose more time attempting to remove them. These problems apply directly to service-based distributed computing systems, in particular clouds.

The RVWI framework addresses the lack of information in autonomic service-based distributed systems by allowing state and attributes of foreign components/services to be shown, thus making it easier for a autonomic distributed system to learn the context of itself, its composing components and any components outside its context. Just like how information about us is important to better improve ourselves, RVWI (through its dynamic WSDL) helps enrich the information that autonomic distributed computing systems have about themselves so that they can further improve their operations and efficiency.

We provided a basic proof of concept of the RVWI Framework by exposing a counter as a web service and demonstrating its changing state via a dynamic WSDL [10, 11]. It was an explicit demonstration of the feasibility of RVWI.

We are currently implementing a large scale proof of concept of RVWI whereby a cluster is to be exposed via a web service which shows the state and attributes of the cluster via the web service's WSDL. We have all our workings in place, all that remains is to pull the cluster and RVWI together and then run management tools, such as schedulers, with the exposed state and attributes. This solution will enrich cloud computing, in particular the Platform as a Service (PaaS) category of clouds.

The main contribution of this work is our RVWI solution, which is an active technology and not only a theoretical model which autonomic computing is mostly driven by. RVWI has previously made major contributions to web services in allowing their state and attributes to be shown via their interfaces without the need for specialised knowledge or tools.

The same contributions are beneficial to autonomic computing as information about itself and any newly discovered components is now easily accessible. Also the exposure of attributes makes it easier to make use of newly discovered components as their operation requirements are now easy to access.

The significance of the project outcome is our framework presents an new foundation for which service-based autonomic distributed computing systems can be built, tested and evaluated. Autonomic computing suffers from the lack of frameworks to help build autonomic systems; the autonomic systems have to be build from scratch. Presenting a new foundation is significant as specialists in the field no longer have to reinvent the wheel.

Our contribution presents a foundation for enriched Self-Awareness and enriched Self-Discovery in service-based autonomic distributed computing systems. While the expose of state and attributes without the need for a specialised service or

component means that information about the autonomic system is freely available and does not have to be extracted. The exposure of attributes improves Self-Discovery as the autonomic system can decide if a newly discovered component or service is usable and demonstrate the required attributes and will not lose time adopting it.

At the next stage of the project we will concentrate our attention on research of a broker and cooperating brokers that will be able to take advantage of the availability of state and attributes published and actuated by services of a autonomic service-based distributed computing system, in particular clouds.

References

1. Amazon. 2007, Amazon Elastic Compute Cloud. <http://www.amazon.com/gp/browse.html?node=201590011>.
2. Amazon. 2007, Simple Storage Service. <http://www.amazon.com/gp/browse.html?node=16427261>.
3. Anthill (University of Bologna, Italy), <http://www.cs.unibo.it/projects/anthill>, (accessed 6 May 2003).
4. D. Box, et al. 2004, Web Services Addressing (WS-Addressing). Updated 10 August 2004. Retrieved 12 September 2007, <http://www.w3.org/Submission/ws-addressing/>.
5. J. Barrera, 1993, <http://www.barrera.org/selftune/selftune.htm>, (accessed 6 May 2003).
6. Bio-inspired Approaches to Autonomous Configuration of Distributed Systems (University College London), <http://www.btexact.com>, (accessed 6 May 2003).
7. T. Bray, et al., 2006, Extensible Markup Language (XML) 1.0, 29 September 2006. <http://www.w3.org/TR/2006/REC-xml-20060816/>.
8. M. Brock and A.M. Goscinski, 2007, 'State Aware WSDL: The Resources Via Web Instances Framework', Technical Report, C07/10, Deakin University, 23 August 2007. <http://www.deakin.edu.au/scitech/sit/dsapp/index.php>.
9. M. Brock and A. Goscinski. 2008. State Aware WSDL. p. 35-44. Sixth Australasian Symposium on Grid Computing and e-Research (AusGrid 2008). Wollongong, Australia ACM. Research and Practice in Information Technology (CRPIT), vol. 82.
10. M. Brock and A. Goscinski, 2007, 'Adding Support for Dynamic State Changes in State Aware WSDL', Technical Report, C07/13, Deakin University, 2 October 2007. <http://www.deakin.edu.au/scitech/sit/dsapp/index.php>.
11. M. Brock and A. Goscinski. 2008. Publishing Dynamic State Changes of Resources Through State Aware WSDL. International Conference on Web Services 2008 (ICWS2008). Beijing, September 2008.
12. D. Bryan, et al. 2002, Universal Discovery, Description, Integration. Updated 19 July 2002. <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>.
13. D. M. Chess, et al. 2004. Unity: experiences with a prototype autonomic computing system. p. 140-147. International Conference on. Autonomic Computing, 2004.
14. E. Christensen, et al. 2001, Web Services Description Language (WSDL) Version 1.1. Updated 15 March 2001. <http://www.w3.org/TR/wsdl>.
15. K. Czajkowski, et al. 2004, From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution. <http://www.globus.org/wsrfl>
16. K. Czajkowski, et al., 2004, The WS-Resource Framework, 5 March 2004. <http://www.globus.org/wsrfl/specs/ws-wsrf.pdf>.
17. I. Foster. 2005. Globus Toolkit Version 4: Software for Service-Oriented Systems. p. 2-13. FIP International Conference on Network and Parallel Computing. Springer-Verlag LNCS 3779.

18. Globus. 2006, Information Services (MDS): Key Concepts. Retrieved 1 October, 2007, <http://www.globus.org/toolkit/docs/4.0/info/key-index.html>.
19. A. Goscinski, 1991, Distributed Operating Systems. The Logical Design, Addison Wesley.
20. A. Goscinski, 2000. 'Towards an Operating System Managing Parallelism of Computing on Clusters of Workstations', *Future Generation Computer Systems*, 293-314.
21. A. Goscinski, et al. 2002. GENESIS: An Efficient, Transparent and Easy to Use Cluster Operating System. *Parallel Computing*, Vol. 28 (2002), No. 4, April, 557-606.
22. A. Goscinski, M. Hobbs and J. Silcock, 2004. Cluster Operating System Support for Parallel Autonomic Computing, Proceedings of the First International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-1), Held in conjunction with 2004 ACM International Conference on Supercomputing (ICS'04), Saint-Malo, France, June 26
23. S. Graham, et al., 2006, WS-BaseNotification, OASIS Specification, 1 October 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.
24. IBM. 2006. An Architectural Blueprint for Autonomic Computing. <http://www-306.ibm.com/autonomic/pdfs/ACwpFinal.pdf>.
25. Immunocomputing (International Solvay Institutes for Physics and Chemistry, Belgium), <http://solvayins.ulb.ac.be/fixe/ProjImmune.html>, (accessed 6 May 2003)
26. P. Horn, 2001, Autonomic computing: IBM's Perspective on the State of Information Technology, Technical Paper, IBM, October 2001. <http://www.ibm.com/developerworks/autonomic/library/ac-summary/ac-manifest.html>.
27. C. M. MacKenzie, et al. 2006, Reference Model for Service Oriented Architecture 1.0. Retrieved November 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.
28. R. March, 2004, Autonomic Computing. On Demand Series, IBM Press. 013144025X.
29. M. Messig and A. Goscinski. 2008. Service Migration in Autonomic Service Oriented Grids. AusGrid 2008. Wollongong, Australia. Proceedings of the 6th Australian Symposium on Grid Computing and e-Research (AusGrid 2008).
30. M. Messig and A. Goscinski, 2007, Autonomic system management in mobile grid environments, in Proceedings of the fifth Australasian symposium on ACSW frontiers - Volume 68. Australian Computer Society, Inc.: Ballarat, Australia.
31. N. Mitra and Y. Lafon, 2007, SOAP Version 1.2 Primer, 27 April 2007. <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>.
32. Multiagent Systems (Freiburg University), <http://www.iig.uni-freiburg.de/~eymann/publications/index.html>, (accessed 6 May 2003).
33. Neuromation (Edinburgh University), <http://www.neuromation.com/>, (accessed 6 May 2003).
34. Nimrod-G (Monash University), <http://www.gridbus.org>, (accessed 28 July 2008).
35. OASIS. 2006, Web Services Resource 1.2 (WS-Resource). Updated 1 April 2006. Retrieved 12 September 2007, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
36. OceanStore (Berkeley University of California), <http://oceanstore.cs.berkeley.edu>, (accessed 28 July 2008)
37. Recovery-Oriented Computing (Berkeley/Stanford), <http://roc.cs.berkeley.edu>, (accessed 28 July 2008).
38. M. Salehie and L. Tahvildari, 2005, 'Autonomic computing: emerging trends and open problems'. SIGSOFT Softw. Eng. Notes, Volume 30, Issue 4, pp. 40-47.
39. B. Sinclair, et al., 2005, Enhancing UDDI for Grid Service Discovery by Using Dynamic Parameters, ICCSA 2005. Springer Berlin: Heidelberg. pp. 49-59. <http://www.springerlink.com/content/mqxvph021tcuxv3m>.
40. R. Sterritt. 2002. Towards autonomic computing: effective event management. p. 40-47. Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE.
41. A. S. Tanenbaum and M. v. Steen, 2002, Distributed Systems: Prentice Hall. 0-13-088893-1.
42. S. Tuecke, et al., 2003, 'Open Grid Services Infrastructure (OGSI) Version 1.0', Draft Recommendation, June 27, 2003.
43. W. Vambenepe, et al., 2006, WS-Topics, 1 October 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.

Bio-inspired Cognitive Radio for Dynamic Spectrum Access

Giacomo Oliveri, Marina Ottonello, and Carlo S. Regazzoni

Abstract Dynamic spectrum access (DSA) has raised the attention of industrial and academic researchers due to the fact that it is seen as a technology able to overcome the lack of available spectrum for new communication services. In particular, autonomic DSA (ADSA) systems are indicated as a solution to spectrum scarcity caused by the current “command and control” allocation paradigm. However, ADSA requires a higher level of reconfigurability with respect to traditional wireless systems. In this context, one of the technologies that can provide such flexibility is the promising cognitive radio (CR). In an ADSA scenario, CR should sense the spectrum to find the resources unused by primary (licensed) users, which could then be exploited by secondary (unlicensed) CR users to increase the overall system efficiency. In this chapter, a comprehensive overview of CR applications to ADSA is carried out; in particular, attention is paid to the potentialities of autonomic bio-inspired approaches, and on their advantages in the solution of the challenges of ADSA systems.

1 Introduction

In the past few years computing systems have evolved to be fully developed and efficient. Of course, there is always a trade-off between efficiency and complexity: in many cases, modern systems have become complex to install, configure, and manage even for skilled users.

Autonomic computing has been proposed to overcome such problems, and at present, it represents one of the most promising topics in computer science. Autonomic computing technologies are designed with the objective of carrying out self-configuration and self-management. Such features appear absolutely necessary for many kind of different and heterogeneous systems, as an example for an autonomic management of communication networks or for software engineering.

G. Oliveri (✉)

Department of Biophysical and Electronic Engineering, University of Genova, Via Opera Pia 11a, 16145 Genova, Italy

e-mail: giacomo.oliveri@dibe.unige.it

Among the possible various applications, autonomic computing can be useful for dynamic spectrum access (DSA). DSA is a promising technology which tries to obtain a more flexible and efficient access to the (shared) spectrum. In this chapter, an autonomic approach to DSA (ADSA) is introduced and the main advantages of such a technique are presented. In particular, the capabilities of self-awareness and self-adaptation, which are highly recommended in a changeable environment as DSA scenario, are considered.

Many problems arise in order to guarantee such a flexibility in ADSA terminal and to solve the challenges introduced by autonomic computing: cognitive radio (CR) technology can be a reasonable answer to realize an adaptive and unsupervised access to the shared spectrum. According to autonomic systems, which can be considered inspired by their biological equivalent, in the chapter bio-inspired CR solutions to DSA will be proposed.

The goal of this chapter is to present an overview of complex and wide topics such as DSA and CR from an autonomic computing point of view. In particular, an autonomic approach to DSA will be considered while CR approaches are presented in order to provide a flexible and adaptive solution to the main issues which arise in ADSA. The focus lays on bio-inspired solutions, which are among the most interesting approaches to CR and represent principles of inspiration for autonomic computing systems.

In order to show the effectiveness of the proposed solution to an autonomic DSA scenario, a bio-inspired cognitive radio approach based on reinforcement learning (RL) algorithms is chosen. RL, a broadly applicable technique in autonomic computing, guarantee to ADSA terminals the capability to learn online also in an unknown scenario without the help of models created by human users. This kind of solution gathers many features demanded by autonomic computing: the proposed system carries out a self-configuration and a self-optimization of its parameters, depending on the conditions of the environment.

In order to verify the performances of the proposed approach in a practical scenario, a bio-inspired cognitive engine is designed and simulated for a vehicular application. In particular, a cognitive base station is implemented by following a RL approach. Simulation result are provided in order to verify the effectiveness of the proposed method, and the capability of the designed system to provide reliable performances in the management of the degrees of freedom of the multiple antenna is shown.

2 Dynamic Spectrum Access

DSA is an emerging technology in the world of wireless communications, and a lot of attention is at present focused on the possibility of exploiting such approach in order to increase the utilization of the radio resource [59]. Such an interest has been driven by the recently published measurements and reports by the Federal Communication Commission (FCC) which show that a great part of the wireless resources, although licensed, are often underutilized [53, 54].

DSA technology is based on the concept that a more flexible wireless access policy can allow a more efficient management of the radio resources: in practice, the objective of DSA is the improvement of the utilization of the spectrum.

In the present section, a description of the basic principles of DSA approach is carried out, along with an overview of the most important problems and challenges in this emerging technology. Then, a distinction among autonomic and nonautonomic approaches in DSA is highlighted.

2.1 Description, Problems, and Challenges in DSA

From a general point of view, DSA can be defined as “a new paradigm of spectrum management, a shift from static allocation to dynamic access” [60]. In practice, the aim of DSA techniques is to overcome the traditional “command and control” approach to the allocation of the radio resources, by allowing a more flexible access to the wireless spectrum [60].

Different strategies can be applied in order to make the traditional spectrum allocation more agile. Such strategies can be grouped in three main categories depending on the considered DSA model: Spectrum Property Rights Model, Open Sharing Model, and Hierarchical Model [59].

In the Spectrum Property Rights Model [36, 59], the free or underutilized resources can be dynamically bought and sold by the license holder, depending on their users’ requests. Even though the Spectrum Property Rights Model introduces flexibility in spectrum management, white spaces resulting from the bursty nature of wireless traffic are difficult to be eliminated through this technique [59].

An alternative is represented by the Open Sharing Model (also named Spectrum Commons Model [36, 59]). In practice, in Open Sharing Models a “lightly controlled” shared access is performed [36]. This model includes, for example, the approach used in the industrial, scientific, and medical (ISM) band [59]. The risk of using this strategy is mainly related to the possible overuse of common resources [36].

Hierarchical Access Model represents the most advanced approach to DSA among those considered here [59]. In this approach, the concept of *primary* and *secondary* terminals is introduced [9]. Primary terminals are the licensed users of the considered radio resource (e.g. frequency channel, time slot, code); secondary terminals, on the contrary, are represented by users that are only allowed to access the considered resource if no primary terminal is going to exploit it [9, 59]. Among the models considered here, the Hierarchical Access Model is perhaps the one with the highest compatibility with the current spectrum management policies and legacy wireless systems [59].

In practice, DSA approaches based on the Hierarchical Model can be considered as wireless systems which have to perform the following operations:

- define when a resource can be considered free or underutilized;
- find the underutilized resources;
- exploit, as “better” as possible, the identified resources.

In general, such tasks are not trivial: a more detailed description of the possible approaches to design a Hierarchical DSA is provided in the following sections.

2.2 Nonautonomic and Autonomic DSA

Many techniques have been proposed in the literature to perform the tasks of a Hierarchical DSA. Among the available subdivisions, it is possible to distinguish between the *autonomic* and the *nonautonomic* approaches.

Autonomic Computing is defined as the set of “information processing and networking technologies that are capable of self-awareness for the purposes of self-optimization, self-healing and self-protection” [16]. This capability of increasing “autonomy and performance by enabling systems to adapt to changing circumstances” [56], which is a basic concept of Autonomic Computing, can assure to DSA self-management and self-adaptation features which cannot be warranted by nonautonomic strategies [16]. In an autonomic context, a DSA radio system will be able to adjust itself to allow high flexibility to dynamic and unexpected situations; that is not feasible in a nonautonomic context, where the user has to manually configure the parameters of the DSA terminal to guarantee the best configuration [31]. As the focus of this chapter is on Hierarchical DSA strategies, the advantages of autonomic and nonautonomic approaches in these models can be considered: in particular, Spectrum Underlay [59], Spectrum Overlay [59], and Spectrum Interweave [55] strategies will be discussed.

Basically, the Spectrum Underlay technique [58, 59] consists in limiting the interferences perceived by the primary users by employing a “mask” which bounds the power transmitted by secondary users and consequently the “interference temperature” [27] present on the channel. The Spectrum Overlay method instead increases the efficiency of utilization of the primary channel by exploiting “interference reduction and cancellation” strategies [13, 25]. Finally, the Spectrum Interweave approach introduces the concept of “opportunity” [55]: a transmission between two secondary users can be performed if a free resource is discovered (e.g spectrum holes [27] or white spaces [24]).

In the Spectrum Interweave model, the concept of opportunity is actually related to the access technology of the primary users. As a consequence, different types of opportunity can be defined: a list of the simplest types of opportunity is reported in Table 1. Generally speaking, the detection and the exploitation of opportunity in the Interweave approach is not a trivial task. Three main subtasks can be identified in this context. Firstly, an opportunistic secondary terminal has to perform

Table 1 Examples of simple types of opportunity as a function of the primary access technology

Primary access technology	Type of opportunity	Examples
FDMA	Frequency opportunity (spectrum hole)	[9, 27]
TDMA	Time opportunity (white space)	[24]
CDMA	Code opportunity	–
SDMA	Space opportunity	–
FDMA/TDMA, CDMA/FDMA	Mixed opportunity	–

an *opportunity prediction* in order to identify the appearance and the length of the opportunity; such prediction has to be confirmed in the subsequent *opportunity detection* phase. Successively, in the *opportunity exploitation*, the secondary terminal has to exploit the (possibly) discovered opportunities, trying to maximize the throughput of the transmission that takes place in the free radio resource. It is important to remark that, in real scenarios, more than two secondary users can try to access the shared spectral resource, in a cooperative or in competitive way [2]: in this case a fourth phase, that is *opportunity sharing*, could be necessary.

While the application of the autonomic approach to Overlay and Underlay communication models does not provide significant advantages, since such applications do not necessarily require adaptive reconfiguration phases [55, 59], autonomic approach is well suited to the Interweave techniques. In fact, all the operations performed by an Interweave radio can be efficiently executed in an autonomic scenario: the main advantage of such choice is, of course, that no direct human operation is required in order to reconfigure secondary terminals for exploiting the opportunity. Thus, only such kind of ADSA systems will be considered in the following.

2.3 Overcoming Flexibility Problems in ADSA: Cognitive Radio

Although Interweave ADSA represents a promising technology, several problems should be overcome in order to allow the design of reliable applications based on this concept. As an example, the level of flexibility demanded for an efficient interweave ADSA could require processing techniques usually not considered in wireless applications.

In order to clarify this point, the simple case of a primary transmission based on a single TDMA channel can be considered. Let us assume that the duration of the “transmission slot” for each primary terminal depends on the kind of data which are transmitted, e.g., short slots for voice traffic, long slots for data traffic (see Fig. 1). Moreover, let us assume that the prevailing traffic type changes periodically (e.g. voice traffic prevails in the evenings and in the weekends).

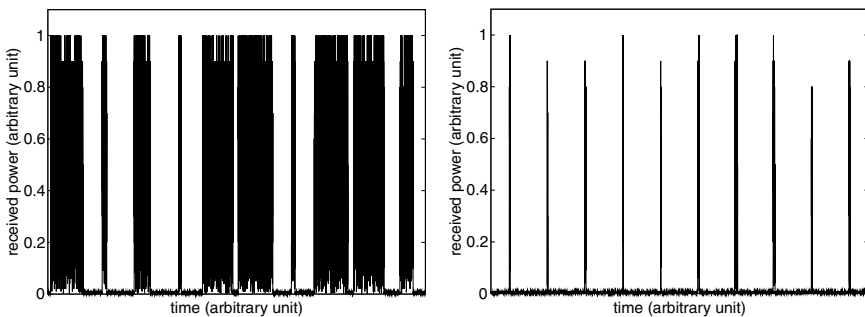


Fig. 1 Power received by the ADSA system in the TDMA example considered in the text. The same arbitrary units are considered in both figures. As it can be seen, different traffic types can lead to different expected duration of the opportunities. *Left*: data traffic. *Right*: voice traffic

In this scenario, an efficient interweave ADSA should consider a lot of different parameters before performing an opportunity prediction (i.e. establish the duration of the white spaces), and its behavior should change in a flexible way in response to environmental changes. In more realistic scenarios, moreover, it could be difficult to a priori establish the parameters to be considered for an efficient reconfiguration algorithm.

The above example shows that the exploitation of technologies able to overcome such kind of flexibility issues could provide a great advantage in interweave ADSA systems. Due to its properties, the CR technology and its application to interweave ADSA will be considered in the following.

3 Cognitive Radio for Autonomic Dynamic Spectrum Access

In the past few years, cognitive systems [28] have attracted the attention of a large number of researchers in the field of communication engineering due to their innovative and appealing properties. Such an interest is confirmed by the number of conferences [14, 15], special issues of international journals [12, 30, 38], books [6, 23, 45], and international projects [2, 35] on this topic.

One of the most promising applications of the cognitive paradigm in communication engineering is represented by the so-called CR [43–46]. At present, CR is already one of the most important emerging technologies in the field of wireless communications [46], and it is seen as fundamental for next generation wireless communications [27]. The great interest in CR paradigm has also recently led to standardization projects [32–35], and such paradigm is already exploited even from a commercial point of view [1, 52].

In this section, an introduction to the concept and to the historical background of CR is provided. Afterward, the application of the CR paradigm to the case of ADSA is discussed, and the advantages of such approach in this context are analyzed.

3.1 Introduction and Motivation

Cognitive systems in communication engineering are defined as flexible and dynamical communication systems that are able to learn from the environment and to provide adaptive and customized services to mobile users [27, 28].

In the previous years several researchers have shown the advantages of cognitive approaches to communications, as an example in the context of flexible communications [45], intelligent routing [6], adaptive radar [29], and smart video-surveillance systems [21]. Such advantages are generally related to the adaptivity and flexibility guaranteed by cognitive approaches with respect to more classical approaches [27, 28]: as a consequence, cognitive approaches are particularly successful when applied to systems which are required to provide reliable performances even in unknown scenarios [27]. This is often the case when emergency-ready communication systems are of interest, both in military and in civilian applications [27].

One of the most promising applications of the cognitive paradigm in communication engineering is represented by the so-called CR [43–46]. From an historical point of view, CR was introduced in 1999 by Mitola and Maguire [46] as an extension of the previously defined software radio [44]. Software radio can be defined as [48]

a radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software.

One of the fundamental issues which suggested the introduction of CR was the automatic, adaptive, and optimized management of the degrees of freedom of software radio platforms, which were starting to be available at low costs [41–44].

Although the flexibility guaranteed by software radio is obviously a benefit from several viewpoints, it can result in an increased complexity of management for the user. CR, therefore, was considered as a way to transform “radio nodes from blind executors of predefined protocols to radio-domain-aware intelligent agents that search out ways to deliver the services the user wants even if that user does not know how to obtain them” [46].

More recently, the CR concept has been widely extended in order to include the capability of the wireless terminals to learn from the environment the most successful reconfiguration strategies on the basis of the perceived context. At present, one of the most widely accepted definitions of CR is that given by Haykin in [27], that is

Cognitive radio is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and adapt its internal states to statistical variations in the incoming RF stimuli by making corresponding changes in certain operating parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in mind:

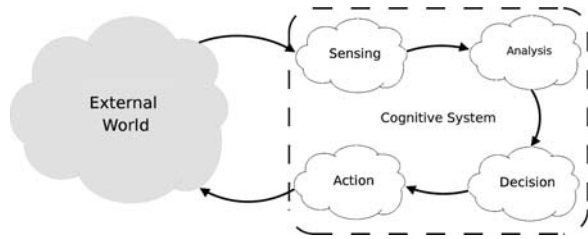
- highly reliable communications whenever and wherever needed;
- efficient utilization of the radio spectrum.

As it is clear from the above definition, the most fundamental characteristics of CR are flexibility and awareness. Such capabilities can be obtained in several different ways, depending on the specific application or context: in this sense, CR represents a converging theme for many different research topics such as signal processing, game theory, and machine learning [27].

From a general viewpoint, the CR approach can be often described by the *Cognitive Cycle* [27, 45]. Different cognitive cycles can be defined depending on the considered application: the cognitive cycle which will be considered here is reported in Fig. 2 [7].

The cognitive cycle represents the internal model of the CR behavior [7, 27]. In practice, a CR can be thought as a system which continuously performs such cycle, if necessary at different levels of the processing stack. The following four tasks define the considered cognitive cycle:

Fig. 2 The considered cognitive cycle, which includes the most fundamental phases of a cognitive radio: *sensing*, *analysis*, *decision*, and *action*. Such cycle is continuously performed in the different levels of the system



- *sensing*, which represents the phase in which the CR collects information from the surrounding environment in the form of low-level data (i.e., the RF signals) through the use of its “body” (i.e. the antenna);
- *analysis*, in which the CR processes the incoming information in order to extract a higher level representation of the context state (i.e., the channel state information);
- *decision*, in which the perceived information is used, together with the available experience (collected during operation and/or provided by the system designer or by the user) in order to establish a new configuration for the system (i.e., the new transmitted power and/or carrier frequency);
- *action*, in which the system applies the decided configuration, interacting with its body and with the surrounding environment.

As it is clear from the above description, the interaction of the “body” of the CR with the surrounding environment plays a central role in the development of the cognitive capabilities: this aspect is detailed in Sect. 4.

As a final observation, it is worth noting that the CR approach does not only apply to the solution of known problems, but also it can provide a strategy to adaptively identify the problem itself [49].

3.2 Examples of Cognitive Radio Approaches to ADSA

In order to describe in more detail the application of a possible CR approach in ADSA, let us consider the practical scenario reported in Fig. 3.

In the proposed scenario, an ADSA application is considered in which a primary wireless service coexists with several secondary terminals in a given band. Such band is assumed to be accessed (by the primary terminals) through FDMA/TDMA: an opportunity is therefore represented by a time–frequency slot. The objective of the ADSA terminal is to detect, predict and, when possible, use the available opportunities without interfering with primary transmissions.

From the description in Sect. 3.1 it can be deduced that the cognitive cycle can allow a straightforward implementation of an ADSA applications: in fact, each of the four phases of the cognitive cycle can be mapped to the ADSA problem in Fig. 3 as follows.

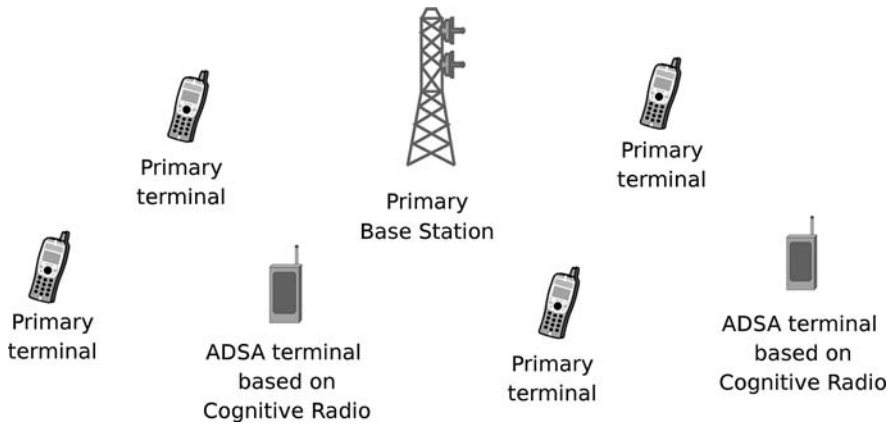


Fig. 3 Practical scenario considered for the application of Cognitive Radio in an ADSA problem. In the considered example, a primary wireless service and several ADSA terminals coexist in the same band

In ADSA, the *sensing* is represented by the phase in which radio signals are perceived by the RF front end. In practice, the result of a sensing phase is the sampled RF signal in the frequencies of interest.

Analysis in ADSA systems can be considered as the phase in which raw RF data are processed and a high-level description of the surrounding wireless context is obtained. In particular, analysis is responsible for extracting information regarding the instantaneous channel occupation on the basis of the perceived signals.

Decision, which is the most complex and important phase in ADSA, is responsible for exploiting the previously extracted information in order to deduce the presence/absence of a time–frequency opportunity. Once detection is performed, such phase has also to decide whether and how (e.g., what power and modulation type shall be used) to exploit the opportunity, and to predict its length. As the duration of an opportunity depends on the considered primary transmissions, experience (e.g. statistics provided by the system designer, or by autonomously learned reasoning) is essential for obtaining reliable predictions.

Finally, *action* in ADSA is responsible for translating the outcome of the decision phase into an actual operation. In the considered example, action will modify the carrier frequency, the output power and the modulation type in order to exploit the opportunity (when detected).

From a practical viewpoint, no limitation is enforced by the CR approach on the various algorithms that could be exploited in the different phases. However, the level of flexibility and adaptivity guaranteed by each part of the system will affect the overall performances. For example, as far as analysis is of interest, several algorithms have been proposed for the detection and classification of narrowband or wideband signals (see, e.g., [20]). As regards decision, in Sect. 4 some bio-inspired algorithms that can be exploited in such phase will be described.

3.3 Benefits and Drawbacks of Cognitive Radio Approach to ADSA

The CR approach allows the realization of effective applications in an ADSA scenario since it is based on the concepts of collection, analysis, memorization and exploitation of the experience in an autonomic way.

The adaptation capability in a distributed and autonomic way is of fundamental importance in ADSA. As an example, since the detection of the opportunities has to be based only on the perceived activity of the primary transmitters and not on a “command-and-control” strategy, the capability of CR to develop autonomic and distributed knowledge is a key advantage over alternative approaches. In fact, such knowledge can be used to overcome the limits of the sensing and analysis phases (i.e. false or missed detections), and to predict some complex characteristics of the primary transmission, such as the expected duration of the opportunity.

Among the advantages that CR approach could provide to ADSA applications, it is possible to cite the following:

- CR approach can allow the development of more robust opportunity detection algorithms, since it can overcome detection errors by using the acquired experience;
- the utilization of the cognitive cycle can help in the definition of macro-functionalities which can be realized without requiring a detailed knowledge of the other parts of the artificial system; in such cases, moreover, the exploitation of the “cognitive cycle” approach can be applied to each subpart (or *agent*) composing the cognitive system; such choice can improve the robustness of the system to isolated failures of its subparts;
- in an emergency-ready ADSA scenario, the tools provided by CR can significantly improve the capabilities of the terminals to efficiently use the available spectrum during or after a network disaster; in such cases, in fact, the cognitive system can perceive a significant change in the behavior of the environment by observing the differences of the perceived situations with respect to the acquired experience, and can therefore exploit innovative solutions (such as employing a larger portion of the spectrum).

Although the above advantages are fundamental, it is important to remark that a flexible and autonomic system can result in potentially dangerous behaviors in wireless applications if a not fully coherent control is enforced on the terminal itself. In fact, suppose that an ADSA CR system is allowed to access the wireless spectrum without any explicit policy or rule (e.g. authentication procedures). Moreover, suppose that the cognitive terminals collect and use experience in an environment which is heavily corrupted by noise. In this case, primary transmissions could be easily (and unintentionally) affected by secondary ones, due to the presence of errors in their acquired experience. This could be particularly dangerous in “protected” bands such as military and satellite communication bands. In this sense, a trade-off between flexibility of the application and control over the guaranteed quality of service for primary users is required in ADSA CR [8].

Among the difficulties that might be overcome when applying a CR approach to ADSA, it is possible to cite that

- flexibility can result in a low level of control over the cognitive system: in practical applications, ad hoc policies and protocols should be adopted to limit the degrees of freedom of ADSA systems;
- Cognitive systems require, in general, a training phase, which may have to be performed (off-line or online, depending on the considered application) before the system is able to perform reliably;
- the exploitation of collected experience could render any terminal virtually unique; as a consequence, identical ADSA terminals could provide different performances: this could be perceived as a low level of reliability, in particular from the point of view of secondary users.

Such disadvantages can be mitigated or overcome by using suitable techniques such as exploiting “distributed decision” algorithms (i.e., algorithms that overcome the limitations of each terminal experience by using cooperation, for example) or enforcing a certain level of control (e.g., based on external policies or on the utilization of reliable experience) on ADSA terminals.

4 Bio-inspired Cognitive Radio Approaches to ADSA

As stated above, CR represents a general framework in which different approaches have been proposed. In this context, “bio-inspired” techniques are among the most interesting and flexible approaches to CR [7, 50], and they have already led to successful applications based, for example, on reinforcement learning techniques, genetic algorithms, or neural networks [4, 7, 37].

In this section the motivations behind the introduction of such strategies are clarified, and a brief description of the most interesting approaches is provided. The advantages of the application of such strategies to an ADSA context are shown. Finally, an overview of bio inspired approaches for the solution of problems related to interwave communications is reported.

4.1 Main Features of Bio-inspired Approaches

Generally speaking, the aim of bio-inspired approaches is “to draw inspiration from biology to introduce novel design guidelines for systems able to show an autonomic behavior” [39]. For these reasons, biologically inspired approaches have recently attracted considerable attention, in particular for applications where the capabilities to adapt and evolve are required, such as robotics [47], cognitive wireless networks [50], and autonomic computing [16].

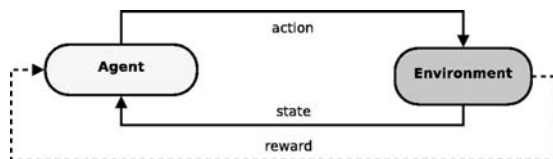
Many heterogeneous approaches to the engineering of artificial cognitive systems inspired by biology have developed in the previous years [11]. Among the most successful ones, it is possible to cite artificial neural networks (ANNs), which can be considered as computational models inspired by the nervous system [11], evolutionary algorithms that are motivated by evolutionary biology [18], and swarm intelligence, which is based on the collective behavior of the social organism [18]. In the following, some particular bio-inspired techniques, i.e., genetic algorithms and reinforcement learning, are considered due to their importance in the field of ADSA systems.

Genetic algorithms, included in evolutionary algorithms [11], are a family of computational models based on the process of natural selection [26, 39]. In such algorithms, the basic concept is the utilization of a *population* of individual entities, which generate new populations through genetic operators such as random mutation, crossover, and selection of the best individuals [26, 40]. The “quality” of each individual is evaluated by using a *functional* depending on the considered problem [26, 40]. By using this concept, genetic algorithms can easily find the optimal solution to complex nonlinear problems, avoiding local minima through suitable genetic operators [26, 40]: for example, recent applications of genetic algorithms in wireless systems include optimized organization of wireless sensor networks [19] and ADSA applications [7, 37, 51]. Although such algorithms are often used as function optimizers, they are well suited also to learning tasks [26, 40].

Reinforcement learning (RL) is a machine learning technique which is focused on “learning by interacting” with the environment [40, 57] (see Fig. 4). Such approach tries to imitate the nature of human learning by exploiting the concept of *reward* [57]. In practice, the basic idea is to perform a “trial-and-error” strategy in order to build a sufficient knowledge of the surrounding environment [57]. The target of the “agent” is to build a *policy* which, given the present state, chooses the action which will probably yield the highest reward (usually in the long run) [57]. Unlike most of analogous machine learning approaches, which are supervised (i.e. learn by examples provided by an external supervisor), “reinforcement learning is learning what to do—how to map situation to actions—so as to maximize a numerical reward signal” [57]. Moreover, under appropriate hypothesis, RL algorithms can guarantee the convergence to the best policy [57].

Three main techniques have been proposed in the literature for the design of effective RL agents: dynamic programming, Monte Carlo methods, and temporal-difference (TD) learning [57]. The TD method includes several different strategies; due to its importance in the following sections, Q-learning method [57] is considered in particular here.

Fig. 4 The agent–environment interaction in reinforcement learning. Such interaction is the basis of the development of intelligence



In order to briefly describe Q-learning, let us provide some notation [57]. Let $s_i \in X$ be the environment state in the instant t_i . Let $a_i \in A$ be the action taken by the agent in t_i , and let $r_i \in \mathbb{R}$ and s_{i+1} be the resulting reward and next environment state. Let R be the “return” that have to be maximized by the agent (e.g. $R_i = \sum_{k=0}^{\infty} \gamma^k r_{k+i+1}$). The agent’s policy is denoted as π_i , where $\pi_i(s, a)$ is the probability that $a_i = a$ if $s_i = s$. The “action-value function” of the policy π , denoted as $Q^\pi(s, a)$, is the expected return starting from s , taking the action a and thereafter following π :

$$Q^\pi(s, a) = E_\pi \{R_i | s_i = s, a_i = a\}$$

Generally speaking, Q^π is not known, and therefore it has to be estimated in order to perform a suitable decision. Among the different techniques that can be used to estimate the Q function, the one-step Q-learning algorithm exploits the following estimation rule [57]:

$$Q_{n+1}(s_i, a_i) = Q_n(s_i, a_i) + \alpha \left[r_{i+1} + \gamma \max_a Q_n(s_{i+1}, a) - Q_n(s_i, a_i) \right] \quad (1)$$

During operation, the estimation of Q is updated on the basis of the explored state space, which depends on the exploited policy: a good choice [57] can be the so-called ε -greedy policy π^ε , that can be expressed as follows:

$$\pi^\varepsilon(s_i, a_i) = \begin{cases} 1 - \varepsilon & \text{if } a_i = \arg \max_a [Q^\pi(s_i, a)] \\ \varepsilon & \text{else} \end{cases} \quad (2)$$

A practical implementation of a Q-learning algorithm can be therefore based on (1) and (2): in practice, the ε -greedy policy π^ε in (2), which improves as the estimation of Q improves, is used to perform the decision by the RL agent. As it can be seen, Q-learning does not require a model of the surrounding environment (Q function is estimated autonomously by the agent), and it can be implemented online, through incremental computation techniques [57]; such characteristics are of particular interest in ADSA applications, as it will be cleared later.

The recent introduction of the concept of embodied cognition [3, 17] has allowed the development of new bio-inspired approaches which are of particular interest for ADSA systems. Embodied cognition is based on the concept, drawn by biological learning entities, that “perception and representation always occur in the context of, and are therefore structured by, the embodied agent in the course of its ongoing purposeful engagement with the world” [3]. On the basis of this approach, some extensions to classical Q-learning algorithms have been proposed in the context of ADSA CR systems [7]. In particular, such extensions are based on the subdivision of the overall state s in two subparts, which can be considered as “internal” and

“external” states [7]. Such subdivision allows the definition of multiple Q functions that can be exploited in different situations, therefore improving the speed of convergence of the learning method [7].

The application of the above described techniques to complete ADSA CR problems is discussed in Sect. 5, while the utilization of bio-inspired techniques in opportunity detection, exploitation, and sharing is considered in the following.

4.2 Opportunity Detection, Exploitation, and Sharing

It has been shown above that the main subproblems in the interweave approach are the opportunity detection/prediction, exploitation, and sharing among users. Different algorithms have been proposed in order to solve such problems: due to their flexibility, it is of interest to consider here few examples of bio-inspired strategies applied to each of these problems.

Let us consider a frequency opportunity detection problem. In this context, one of the most critical tasks is to classify the modulation technique of the incoming signal(s) [27]. To this end, bio-inspired classification algorithms have been proposed for example in [22] to obtain a more efficient and reliable strategy. In particular, ANNs together with cyclic spectral analysis are considered [22]. The ANNs constitute a highly flexible bio-inspired method that allows to “circumvent issues with classification where the signal’s carrier and bandwidths are unknown” [22]. An efficient bio-inspired approach to detection is also shown in [49]: in this case the system is modeled on the “cognitive cycle” already discussed.

Regarding opportunity exploitation, it is known that such problem is somehow similar to that usually solved by Adaptive Modulation and Coding techniques [10]. However, it is worth remarking that “estimation of the communication performance achievable with respect to environmental factors and configuration parameters plays a key role in the optimization process performed by a Cognitive Radio” [4]. For this reason, a higher level of flexibility may be required with respect to the techniques usually considered in Adaptive Modulation and Coding [4]: as an example, a Multilayered Feedforward Neural Networks (MFNN) has been proposed in [4] to synthesize performance evaluation functions in CRs; furthermore, in [10] a reasoning and a learning engines are employed to maximize the capacity of additive white Gaussian noise (AWGN) and non-AWGN channels.

Finally, as far as opportunity sharing is of interest, a fair spectrum access can be achieved by exploiting cooperative or uncooperative strategies. In particular, “cooperative solutions consider the effect of the node’s communication on other nodes” [2]: this can be obtained by using distributed or centralized bio-inspired approaches [5, 49]. As an example, in [5] a self-synchronization mechanism based on biological systems is used for implementing a global optimal distributed decision system, while in [50] a distributed approach based on swarm intelligence is proposed for the harmonious exploitation of finite spectral resources.

5 Present Applications and Possible Future Scenarios

In the previous sections, the possibilities offered by CR in ADSA have been clarified from several point of views. In particular, the capability to acquire experience from the interaction with the environment has been remarked as fundamental in ADSA applications in order to overcome, for example, the failures of the primary network, of other ADSA terminals, or of subparts of the cognitive cycle.

However, as partially anticipated, some open issues have to be faced in order to allow the definition of feasible and reliable CR systems in ADSA scenarios. In this section, some of these issues (both from a technical and a commercial point of view) are addressed, and a proposed original solution is discussed in detail.

5.1 Research in Bio-inspired Cognitive Radio for ADSA

As far as CR ADSA systems are considered, bio-inspiration represents one of the most interesting approaches, and its advantages have already been remarked above. However, as already stated, different levels of cognition and intelligence may be chosen depending on the considered application. Due to the high number of contributions on this topic in the literature, only some of the most advanced approaches to bio-inspired management of ADSA terminals will be considered here. For a more complete overview of the research in the field, see [12, 38].

As a first example of the current trends in the research community, the capabilities of bio-inspired approaches to manage emergency situations in real time have recently received much attention due to the fact that this represent one of the most interesting application of the cognitive paradigm [37, 51]. Such researches, which partially exploit some of the approaches presented in [27], have been carried out by applying advanced machine learning techniques to the CR problem [37, 51]. In particular, the target of these works is to design and test an overall cognitive system able to [37, 51]

- learn from the interaction with the environment;
- develop and organize experience;
- try “new” solutions when facing unexpected problems.

In order to meet these goals, complex architectures based on extended cognitive cycles and particular bio-inspired techniques have been developed.

As an example, in [37] a system is developed in order to “provide the universal interoperability for public safety communications”: to this end, case-based reasoning based on reinforcement learning is used in order to acquire and exploit experience, while genetic optimizers are chosen for the implementation of creativity in the decision phase [37].

Other approaches that exploit the concept of bio-inspired CR have been recently developed in the field of multiple-antenna ADSA systems. In particular, the focus

of these researches is on the capability of CR-based ADSA to exploit the additional degrees of freedom of multi-antenna systems (with respect to single-antenna systems) through machine learning approaches [7]. In this case, the target is to exploit experience to build a self-trained spatially aware ADSA system, which is capable to steer the available antenna array toward the directions of interest on the basis of the perceived signals [7]. In this way, opportunistic communications can be established between the ADSA system and the surrounding wireless terminals without causing interference to other systems in the domain of interest by exploiting spatial diversity.

The technique chosen in [7] to allow the system to learn from experience is Q-learning, in particular based on the concept of embodied cognition. Moreover, genetic optimizers are used for generating creative solutions if “unexpected” (i.e. not available in the experience database) situations are encountered. A detailed example of a system exploiting the concept of self-trained multi-antenna ADSA will be provided in Sect. 5.2.

A resume of the main bio-inspired techniques exploited in some recent contributions regarding ADSA systems based on CRs is reported in Table 2. As it can be seen, the exploitation of bio-inspired techniques is of fundamental importance in these cases to allow the definition of effective autonomic DSA systems.

5.2 Application: Bio-inspired Cognitive Radio Approach for an Autonomic DSA Exploiting Spatial Opportunities

In this section, a bio-inspired ADSA system is proposed and the design and training phases of an ADSA system exploiting multiple antennas is described.

To this end, some assumptions regarding the considered application are required. In particular, an interweave ADSA scenario is considered in which several mobile primary terminals exploit narrowband transmissions (see Fig. 5).

The ADSA terminal is expected to track the position of the mobile primary users in its vicinity and to try to establish a communication with them without causing interferences. In order to allow the ADSA terminal to exploit spatial diversity, the system is provided with an electronic steerable antenna array. In the considered example, therefore, an opportunity is defined as an established connection with a terminal (i.e. it is represented by a carrier frequency and a direction of communication, for a certain period).

Table 2 Examples of bio-inspired techniques exploited in recent contributions regarding ADSA systems based on cognitive radios

Reference	Applied bio-inspired techniques
[7]	Simple cognitive cycle, embodied-cognition-based reinforcement learning, genetic optimization
[37]	Extended cognitive cycle, reinforcement learning, genetic optimization
[51]	Extended cognitive cycle, genetic optimization

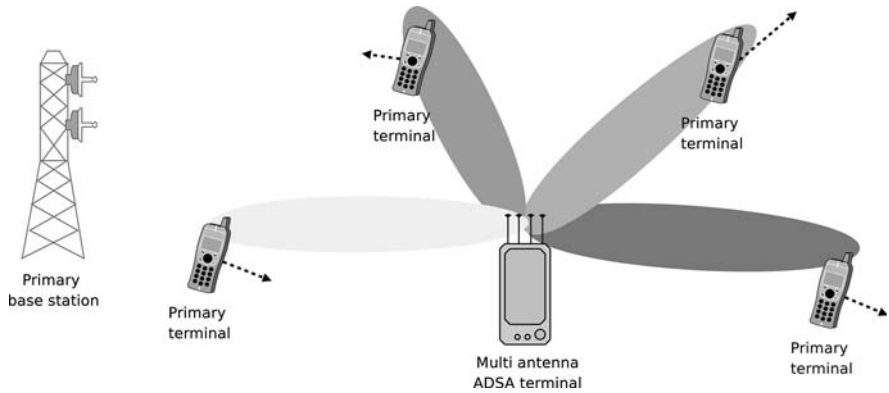


Fig. 5 Considered scenario for the designed ADSA system provided with multiple antennas. Several mobile primary terminals are considered. The main lobe of the hypothetical communications established with the primary terminals are reported (gray tones denote carrier frequencies)

In order to solve the above ADSA problem, a design is proposed on the basis of bio-inspired CR approach. In particular, the following (discrete time) cognitive cycle is proposed for the considered system.

- *Sensing*: the data are collected by the antenna array whose steering direction span a certain number of directions of interest in a given time; in particular, for each “direction of interest” a portion of the signal is captured and transformed through FFT; then, the resulting spectral information is fused to create a “raster scan” of the spectrum in the directions around the ADSA system. In Fig. 6 (left) an example of the “sensing map” obtained by the ADSA terminal is shown. It is easy to note the presence of four energy peak values (in this particular case), corresponding to four mobile terminals in the domain of interest.

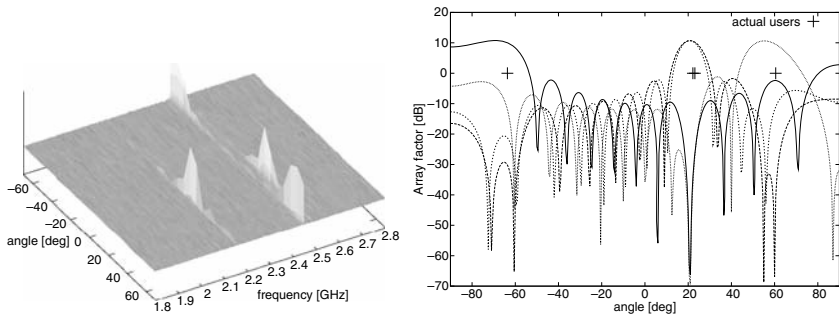


Fig. 6 Example of sensing maps (left) and outcome of the associated action (right). In this particular sensing map, four users at different frequencies and at different directions can be detected (by the analysis phase). This information is used in the action phase, in order to correctly reconfigure the beamformer to minimize the mutual interference

- *Analysis*: the data collected by sensing are processed in order to extract, for each direction of interest, the number of detected terminals and the related carrier frequencies. Since narrowband signals are considered, post-FFT energy detector are used on the sensing maps in order to detect the presence/absence of a terminal. The set of detected terminals and associated frequencies is passed to the decision.
- *Decision*: by exploiting the concept of embodied cognition, the system state is divided into internal (configuration for the attempted communications, i.e. antenna array configuration, used power, carrier frequency for each communication) and external state (detected position/frequency of the terminals, data for the established connections). Effective configuration strategies are learned through operation by exploiting a Q-learning approach [57], in which the reward is a function of the number of established connections and of the transmission power used to establish such connections. In particular, an ϵ -greedy reinforcement learning [57] is chosen, where ϵ decreases as the amount of acquired knowledge increases. A more detailed description of the parameters used in the Q-learning approach for the considered application are shown in Table 3. The outcome of the decision is represented by the set of attempted communications for the following phase.
- *Action*: the configuration for the communications is applied to the subparts of the ADSA system (antenna array, RF stages, etc.) and the outcome of such operation is collected and then passed to the next sensing phase. In Fig. 6 (right) the configuration of the beamformer is shown. In particular, it is worth remarking the presence of the peaks of the radiation pattern corresponding to the four mobile terminals detected in the analysis phase.

The above described phases represent the logical basis for the development of the proposed ADSA management software.

Numerical simulations of the proposed system have been carried out in order to verify the effectiveness of such approach. To this end, a complete C++ simulator has been developed, which includes the management software for the ADSA system and the emulation of the surrounding environment and primary terminals. In the numerical simulations, pedestrian mobility for the primary terminals is supposed, and they are assumed to transmit in the band 1.8–2.8 GHz. The antenna of the ADSA system is a linear equispaced array of 21 dipoles, and the simulated channels include free space losses and additive white Gaussian noise.

In the following, simulation results are reported for the case that at most two primary terminals lie in the domain of interest of the ADSA system in each moment. In the considered example, the system is designed with the objective to try to establish

Table 3 Example of the Q-learning approach for the considered system

Q-learning element	Corresponding in simulator
s_i	Number of detected users, directions associated to the users, SNR of each established link, transmission modality
a_i	Beamformer configuration. power for each link
r_i	SNR (to maximize), transmitted power (to minimize)

the maximum number of connections without considering the amount of power required to perform this task: no additional knowledge is provided to the system. As a consequence, the system is expected to minimize the steering error towards the mobile primary terminals. The results of a training phase of about 450 s are reported in Figs. 7 and 8.

In particular, the absolute steering error $e(t)$ is calculated by using the following equation:

$$e(t) = |\theta_e(t) - \theta_c(t)| \tag{3}$$

in which θ_e denotes the steering direction estimated by the ADSA system while θ_c denotes the correct steering direction (the actual position of the mobile primary

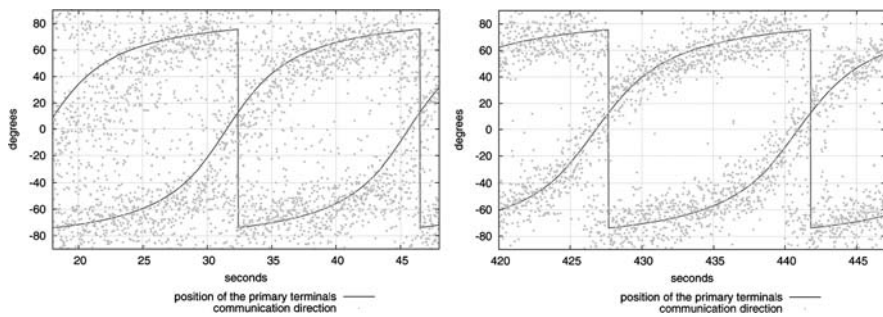


Fig. 7 Example of the tracking performances of the proposed multi-antenna ADSA. In these simulations, at most two terminals lie in the domain of interest in each moment, and each terminal follows a straight trajectory at constant speed. The angular position of each terminal is represented by the lines, while the directions of the attempted communications are represented by the dots

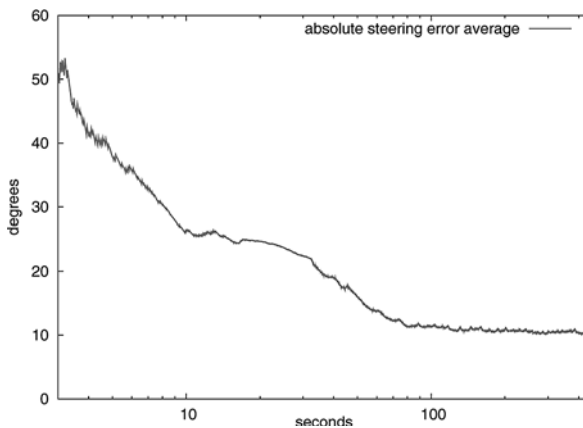


Fig. 8 Evolution of the absolute steering error of the designed multi-antenna ADSA during the training phase. In the considered example, at most two terminals lie in the domain of interest in each moment. The absolute steering error is averaged over $T = 30$ s

terminal) both calculated at the time instant t . The absolute steering error average shown in Fig. 8 is then obtained by averaging the absolute steering error (3) over a sliding window of T seconds, i.e. $e_{\text{mean}} = (1/T) \int_T e(\tau) d\tau$.

As it can be seen from the reported results, the proposed cognitive cycle allows the solution of the considered problem in a flexible way, since the designed ADSA system is able to learn the correct steering strategy in an autonomic way and without requiring the definition of an a priori optimal strategy.

Moreover, from the reported results it is possible to deduce that bio-inspired techniques are able to manage the available degrees of freedom and therefore to exploit “spatial opportunities,” at least in the simple considered scenario. Of course, more complex tests should be carried out in order to establish if such an approach can lead to reliable performances in more realistic scenarios (e.g., considering wideband modulations and multipath channels). However, the obtained results, together with the previous considerations, already suggest that the considered approach can solve such problems in an effective way.

5.3 Practical and Commercial Issues in Bio-Inspired Cognitive Radio Approaches for ADSA

Despite the interesting properties of bio-inspired CRs for ADSA, open issues exist from a technical and a licensing point of view that have to be addressed in order to make such systems more easily usable in a commercial context.

Let us firstly consider a technical issue not discussed in Sect. 3.3. As far as autonomic interweave communications are of interest, spectrum sensing can represent a very difficult task. In fact, let us consider the case represented in Fig. 9. In this example, any attempt to evaluate the occupation of the spectrum would fail, even in the case of cooperating ADSA terminals. The considered problem could even appear

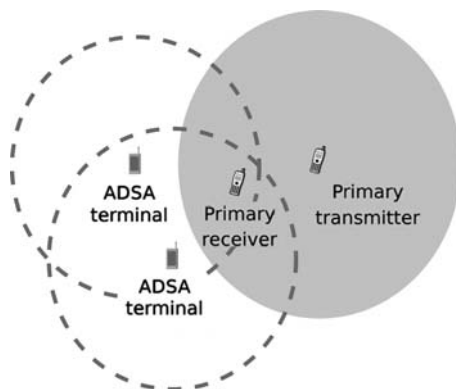


Fig. 9 Example of erroneous opportunity detection. Empty circles represent the transmission ranges of ADSA terminals, while the gray circle represents the transmission range of the primary transmitter. As it can be seen, primary (silent) receiver will be affected by unwanted secondary interference, since both ADSA terminals will always detect an opportunity

in simpler situations, especially if strong multipath is present. Although the considered problem has no simple solution, its importance can be significantly reduced if multi-antenna or distributed detection algorithms [49] are employed, since in this case the probability that a primary transmitter is “invisible” to all ADSA receivers decreases with the number of coordinated or cooperating receivers.

Another important issue in CR for ADSA is related to the intrinsically unpredictable behavior of such systems [8]. Since, in general, “predictable behavior is highly prized in radio systems” [8], ADSA systems based on CR may be required to mitigate their unpredictability in order to be appreciated by users, designers, and regulators [8]. A possible solution to this problem is related to the capability of the ADSA system to guarantee a minimal quality of service and to perform better whenever possible [8].

Furthermore, one of the most complex issues in ADSA based on Cognitive Radio regards the spectrum licensing policy [9]. In fact, since ADSA falls in the category of “noncooperative DSA” [9], several obstacles can limit the deployment of such applications: in particular, such kind of DSA does not only require a change in the regulatory policy, but could even introduce new business models in the wireless industry, making “incumbent business models based on spectrum scarcity less viable” [9]. Although a sudden change in the regulatory and industrial processes is not possible, the gradual introduction of ADSA systems in commercial application is seen as a possible way to overcome such obstacles [9]. The recent and successful efforts in the definition and standardization of DSA applications in the TV band [35] represents a remarkable success in this context.

In conclusion, it is possible to note that challenges in the development of ADSA systems based on bio-inspired CR approaches exist. However, the capability of such approaches to overcome most of the problems related to lack of flexibility in dynamic spectrum access, demonstrated in the present chapter, along with the available promising works in this research field, will certainly represent the most important advantages of such approaches in the definition of tomorrow’s wireless applications.

6 Conclusions

In this chapter a comprehensive overview of CR applications to ADSA has been carried out. After an introduction on DSA principles and challenges, the possible application of Autonomic Computing techniques to DSA has been discussed. Given the objective and constraints of ADSA, the role of the CR approach in the definition of flexible ADSA applications has been remarked. In particular, bio-inspired CRs have been discussed in detail, and reinforcement learning has been introduced as a possible technique to provide autonomic and flexible capabilities to DSA applications.

The overview of ADSA technology has been exploited for the subsequent design of an innovative ADSA application exploiting spatial opportunities. The scenario for the considered application has been discussed, and the control algorithm for the

CR engine which performs the wireless spectrum management has been described in detail. The effectiveness of bio-inspired CR approaches in ADSA has been therefore shown in a practical example. The resulting ADSA system has been validated through software simulations, and the adaptivity guaranteed by the reinforcement learning technique has been tested in the problem of keeping a connection with mobile traveling terminals in a wireless context.

Finally, some practical and commercial issues in bio-inspired CR approaches for ADSA have been reported, and some possible techniques and ideas that could be applied to overcome such problems have been discussed.

References

1. Adapt4 Inc (2008) XG1™ Cognitive Radio. <http://www.adapt4.com/adapt4-products.php>
2. Akyildiz IF, Lee WY, Vuran MC, Mohanty S (2006) NeXt generation/dynamic spectrum access/cognitive radio wireless networks: a survey. *Computer Networks* 50:2127–2159
3. Anderson ML (2003) Embodied cognition: a field guide. *Artificial Intelligence* 149:91–130
4. Baldo N, Zorzi M (2008) Learning and adaptation in cognitive radios using neural networks. In: 5th IEEE Consumer Communications and Networking Conference, pp 998–1003
5. Barbarossa S, Scutari G (2007) Bio-inspired sensor network design. *IEEE Signal Processing Magazine* 44(3):26–35
6. Bhargava KV, Hossain E (2007) *Cognitive Wireless Communication Networks*. Springer, Berlin
7. Bixio L, Oliveri G, Ottonello M, Raffetto M, Regazzoni CS (2007) A reinforcement learning approach to cognitive radio. In: *Software Defined Radio Technical Conference Proceedings*, Denver, USA
8. Chapin JM, Doyle L (2007) A path forwards for cognitive radio research. In: *Second International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Orlando, USA, pp 127–132
9. Chapin JM, Lehr WH (2007) The path to market success for dynamic spectrum access technology. *IEEE Comm Mag* 45(5):96–103
10. Clancy C, Hecker J, Stuntebeck E, O’Shea T (2007) Applications of machine learning to cognitive radio networks. *IEEE Wireless Communications* 14(4):47–52
11. Cliff D (2003) Biologically-inspired computing approaches to cognitive systems: a partial tour of the literature. Tech. Rep. HPL-2003-11, Digital Media Systems Laboratory, HP Laboratories, Bristol
12. Cordeiro C, Daneshrad B, Evans J, Mandayam N, Marshall P, Shankar S (eds) (2007) Special issue on adaptive, spectrum agile, and cognitive wireless networks. *IEEE J Sel Area Comm* 25(3):513–516
13. Costa MHM (1983) Writing on dirty paper. *IEEE Trans Inform Theory* 29(3):439–441
14. CRN Workshop (2008) 2nd IEEE International Workshop on Cognitive Radio Networks. http://cms.comsoc.org/CCNC_2008/Content/Home/Call_for_Papers_/CRN_Workshop.html
15. CROWNCom (2008) International conference on cognitive radio oriented wireless networks and communications. <http://www.crowncom.org/>
16. Cybenko G, Berk VH, Gregorio-De Souza ID, Behre C (2006) Practical autonomic computing. In: *Proceedings of the 30th Annual International Computer Software and Applications Conference*, Washington, DC, USA, pp 3–14
17. Damasio A (1999) *The Feeling of What Happens: Body and Emotion in the Making of Consciousness*. Harcourt Brace, San Diego
18. De Castro LN, Von Zuben FJ (2005) *Recent Developments in Biologically Inspired Computing*. Idea Group Publishing, New York

19. De Mello RF, Cuenca RG, Yang LT (2006) Genetic algorithms applied to organize wireless sensor networks aiming good coverage and redundancy. In: First International Conference on Communications and Networking in China, pp 1–5
20. Dobre O, Abdi A, Bar-Ness Y, Su W (2007) Survey of automatic modulation classification techniques: classical approaches and new trends. *IET Communications* 1(2):137–156
21. Dore A, Pinasco M, Regazzoni CS (2007) A bio-inspired learning approach for the classification of risk zones in a smart space. In: Online Learning for Classification Workshop, Minneapolis, USA, pp 1–8, URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4270438
22. Fehske A, Gaedert J, Reed JH (2005) A new approach to signal classification using spectral correlation and neural networks. In: First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, pp 144–150
23. Fette BA (2006) *Cognitive Radio Technology*. Newnes, Oxford
24. Geirhofer S, Tong L, Sadler BM (2007) Dynamic spectrum access in the time domain: modeling and exploiting white space. *IEEE Comm Mag* 45(5):66–72
25. Han T, Kobayashi K (1981) A new achievable rate region for the interference channel. *IEEE Trans Inform Theory* 27(1):49–60
26. Haupt RL, Haupt SE (2004) *Practical Genetic Algorithms*, 2nd edn. Wiley, New York
27. Haykin S (2005) Cognitive radio: brain-empowered wireless communications. *IEEE J Sel Area Comm* 23(2):201–220
28. Haykin S (2006) Cognitive dynamic systems. *Proceedings of the IEEE* 94(11):1910–1911
29. Haykin S (2006) Cognitive radar: a way of the future. *IEEE Sig Proc Mag* 23(1):30–40
30. Haykin S, Li G, Shafi M (eds) (2008) Special issue on Cognitive Radio, *Proceedings of the IEEE*
31. Ibrahim MT, Anthony RJ, Eymann T (2006) Exploring adaptation & self-adaptation in autonomous computing systems. In: *Proceedings of the 17th International Workshop on Database and Expert Systems Applications*, Los Alamitos, USA, pp 129–138
32. IEEE 80216 License-Exempt (LE) Task Group (2008) Web site. <http://ieee802.org/16/le/>
33. IEEE 80222 Working Group (2008) Web site. <http://www.ieee802.org/22/>
34. IEEE Communications Society TCCN (2008) Web site. <http://www.eecs.ucf.edu/tccn/index.html>
35. IEEE Standards Coordinating Committee 41 (2008) Web site. <http://www.scc41.org/>
36. Ileri O, Mandayam N (2008) Dynamic spectrum access models: Toward an engineering perspective in the spectrum debate. *IEEE Communication Magazine* 46(1):153–160
37. Le B, Garcia P, Chen Q, Li B, Ge F, El Nainay M, Rondeau T, Bostian C (2007) A public safety cognitive radio node system. In: *Software Defined Radio Technical Conference Proceedings*, Denver, USA, URL <http://www.sdrforum.org/SDR08/3.3-2.pdf>
38. Liang YC, Chen HH, Mitola J, Mahonen P, Kohno R, Reed JH (eds) (2008) Special issue on cognitive radio theory and application, vol 26, *IEEE J. Sel. Area Comm.*
39. Miorandi D, Yamamoto L, Dini P (2006) Service evolution in bio-inspired communication systems. *International Transactions on Systems Science and Applications Journal* 2(1): 51–60
40. Mitchell TM (1997) *Machine Learning*. McGraw-Hill, New York
41. Mitola J (1992) Software radios-survey, critical evaluation and future directions. In: *National Telesystems Conference*, pp 13/15–13/23
42. Mitola J (1995) The software radio architecture. *IEEE Comm Mag* 33(5):26–38
43. Mitola J (2000) Cognitive radio: An integrated agent architecture for software defined radio. PhD thesis, Royal Institute of Technology (KTH), Sweden
44. Mitola J (2000) *Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering*. Wiley, New York
45. Mitola J (2006) *Cognitive Radio Architecture: The Engineering Foundations of Radio XML*. Wiley, New York
46. Mitola J, Maguire GQ (1999) Cognitive radio: Making software radios more personal. *IEEE Pers Commun* 6(4):13–18

47. Pfeifer R, Lungarella M, Iida F (2007) Self-organization, embodiment, and biologically inspired robotics. *Science* 318(5853):1088–1093
48. Reed JH (2002) *Software Radio: A Modern Approach to Radio Engineering*. Prentice Hall, New York
49. Regazzoni CS, Gandetto M (2007) Spectrum sensing: a distributed approach for cognitive terminals. *IEEE J Sel Area Comm* 25(3):546–557
50. Renk T, Kloeck C, Burgkhardt D, Jondral FK, Grandblaise D, Gault S, Dunat JC (2007) Bio-inspired algorithms for dynamic resource allocation in cognitive wireless networks. In: *International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Orlando, FL, USA, pp 351–356
51. Rieser CJ (2004) *Biologically inspired cognitive radio engine model utilizing distributed genetic algorithms for secure and robust wireless communications and networking*. Phd thesis, Virginia State University
52. Shared Spectrum Company (2008) Web site. <http://www.sharedspectrum.com/>
53. Spectrum Policy Task Force (2002) Report of the spectrum efficiency working group. Tech. rep., Federal Communications Commission
54. Spectrum Policy Task Force (2002) Report of the spectrum rights and responsibilities working group. Tech. rep., Federal Communications Commission
55. Srinivasa S, Jafar SA (2007) The throughput potential of cognitive radio: a theoretical perspective. *IEEE Comm Mag* 45(5):73–79
56. Sterrit R, Bustard D (2003) Towards an autonomic computing environment. In: *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, Prague, Czech Republic, pp 694–698
57. Sutton RS, Barto AG (1998) *Reinforcement learning*. The MIT Press, Cambridge, Massachusetts
58. Zhang H, Zhou X, Yazdandoost KY, Chlamtac I (2006) Multiple signal waveforms adaptation in cognitive ultra-wideband radio evolution. *IEEE J Sel Area Comm* 24(4):878–884
59. Zhao Q, Sadler BM (2007) A survey of dynamic spectrum access. *IEEE Sig Proc Mag* 24(11):79–89
60. Zhao Q, Tong L, Swami A (2005) Decentralized cognitive mac for dynamic spectrum access. In: *Proceedings of the First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, Baltimore, USA, pp 224–232

Introducing Autonomous Behaviors into IMS-Based Architectures

Mohamed Boucadair

Abstract The main objective of this chapter is to describe a set of viable solutions aiming to enhance the robustness and the availability of current IMS-based architectures owing to the activation of autonomic-like techniques. This chapter adopts a Service Provider standpoint and clarifies the scope of autonomic behaviors within an administrative service domain owned and managed by a Service Provider. The rationale elaborated within this chapter argues in favor of introducing autonomic and distributed means into current operational service platforms in order to build survival and deterministic networks. Autonomic networking means are not used as alternative solutions but as an enhancement to the already deployed ones. For backward compatibility and for migration issues, this chapter recommends to enforce the proposed solutions as backup ones in the earlier stages of deployment. Once field proven, the proposed mechanisms could be enforced as primary procedures to deliver more sophisticated services. This chapter investigates how autonomic networking and more precisely distributed techniques may be implemented and engineered in the context of IP Multimedia Subsystem (IMS)-based architectures. This effort is required so as to disseminate autonomic networking inside Telcos community mainly by proposing concrete and viable solutions which meet operational requirements. This chapter does not require specific background on autonomic networking and computing.

1 Introduction

1.1 General Overview

Telephony over IP has gathered the interest of a large number of researchers and engineers in both standardization organizations and industrial fora focusing on Service Providers issues. Thus, numerous protocols and architectures have been proposed in order to deploy IP-based telephony service offerings. A large part of

M. Boucadair (✉)
France Télécom R&D, 42 Rue des coutures, 14066 Caen Cedex, France
e-mail: mohamed.boucadair@orange-ftgroup.com

these protocols are introduced into operational platforms. Nevertheless, several challenges, such as the migration to IPv6 [6] and inter-working with IPv4 [2, 3, 21], interconnection between Service Providers IP telephony domains [18], service robustness, and availability, security, and QoS (quality of service), are still open issues and should be solved. Moreover, a sensitive challenge regarding VoIP (Voice over IP) architectures is to build lightweight architectures which meet Service Providers requirements. The complexity of the current proposed solutions should be questioned against the added values of invoked functional elements. We believe that a balanced approach between the “flat” approach a la Internet and the one advocated by 3GPP (3rd Generation Partnership Project, [<http://www.3gpp.org>]) is possible and even realistic.

The area of investigation of this chapter is aligned with the big trend of enhancing IP network design and its ability to offer sophisticated conversational services, but not limited to, in a native way, i.e., with no major operational and architectural pains/complexity. Flexibility to leverage innovative services is one of the preoccupations that motivate the proposed architectures described within this chapter. The target of this chapter is not to describe how to duplicate the technical choices practices inside Telcos’ organizations but to exploit innovative facets of autonomic emerging techniques [15], and particularly peer-to-peer (P2P) and distributed-oriented solutions, to enhance the QoS as experienced by end-users.

This chapter adopts a pragmatic and realistic approach. Our position regarding P2P techniques is not evangelistic. Therefore, we assume that some autonomic-like solutions, such as P2P-based solutions, are not the “perfect” answer for all technical issues. We are aware that these solutions suffer from several technical hurdles and do not provide optimal solutions for all encountered problems. To illustrate this weakness, we recall readers about the recent crash of Skype service due to an OS (operating system) update [<http://www.p2pnet.net/story/13137> and <http://www.p2pnet.net/story/13105>]. Indeed, the service was unavailable and the overall system was not able to immediately converge to a stable state allowing end-users to access to Skype services. Another example is the current problems encountered by Joost, an emerging distributed IP TV service offering, to ensure an AS-level (autonomous System) for traffic optimization and the control of the last mile to optimize the delivery of media streams. Recent measurements show that the majority of the traffic is sent by servers owned by Joost and that more than 100 Joost-enabled clients contribute to send media streams but the amount of generated traffic is minor compared to the one sent by Joost servers (for more details about these measurements and additional details refer to [12, 13, 22]).

In order to meet the lightweight and flexibility requirements to deploy conversational services and to benefit from the autonomy characteristics of new emerging autonomic-like applications, Service Providers shall investigate new solutions either starting from the architectures already deployed in their operational networks or radically exploring new tracks. Of course, backward compatibility with the already running service offerings and migration strategies of individual Service Providers should be taken into account when designing these new solutions. Note that the challenge is not only to allow exchange of media streams between users but to

offer telephony services compliant with both regulatory constraints (such as Legal Interception and Emergency Calls) and also with operational directives (e.g., control the experienced quality of the service, ease troubleshooting operations, ease introduction of new services, manage interconnection with adjacent VoIP Service Providers, implement basic billing features). Among all the conceivable options, this chapter focuses only on the one that consists in investigating how current centralized architectures such as IMS (IP Multimedia Subsystem) [4] could evolve, be adapted, and/or enhanced to meet some critical requirements, mainly, flexibility, dynamic failure detection and recovery, autonomous “tuning” to solve overload problems, etc.

The approach adopted within this chapter happens to be realistic and viable since it is suitable for traditional telephony Service Providers. The chapter describes incremental solutions to be adopted so as to ensure backward compatibility and to optimize the required investment both in term of CAPEX (Capital Expenditure) and OPEX (Operational Expenditures). Elaborated migration strategies should be elaborated so as to drive the introduction of such autonomic architectures into operational networks (based on IMS or TISpan [20]). An example of migration strategy could consist in enhancing the current IMS architectures with the introduction of ad hoc distributed functions at the access segments in order to improve the resilience of the service offerings. Once these capabilities would be supported by the access segment elements, some core service functions could be delegated to these elements and then deploy a fully dynamic and distributed system providing similar services. Concretely, and because the author of this chapter is aware about operational constraints such as backward compatibility, we adopt an incremental approach.

The solutions introduced in this chapter are inspired from the capabilities of the newly promoted autonomic paradigms and context-awareness models. Once field proven, these mechanisms can be elected to be used as primary modes to deliver service offerings. The deployment of the proposed solutions as primary solutions is not described in this chapter. In addition, introducing autonomic networking techniques inside operational networks should be incremental. Techniques for dynamic provisioning and service automation should be promoted within Service Providers community. Besides this concern, self-care techniques and service sanity checking means should be proposed so as to meet the requirements of Service Providers. The aim of this chapter is to contribute to the dissemination of autonomic networking from the standpoint of a Service Provider.

Autonomic or autonomous networking are generally presented as self-* solutions. Within this chapter, only availability and robustness characteristics are taken into account. In the remaining part of this chapter, P2P, distributed solutions, and autonomic networking terms are used interchangeably.

1.2 This Chapter at a Glance

The main focus of this chapter is to describe solutions aiming to enhance the robustness and the availability of current IMS-based service offerings owing to the

activation of techniques inspired from autonomic networking. These solutions are not software-based but are network-centric and are inspired from native IP techniques. These solutions are said to belong to “autonomic networking” because, at an abstraction level, autonomic concepts are followed.

Several failure vulnerabilities may be identified and then investigated in depth. This chapter encloses a description of autonomic and fully distributed modes to solve some of these failure scenarios. Hereafter we provide the three failure types which are within the scope of this chapter:

- *Failure of access nodes*: This chapter describes a lightweight approach based on appropriate engineering practices so as to prevent against failures and to provide the service even in case of failure of access nodes. The proposed mechanism is transparent to end-users. This mode does not require any involvement of humans.
- *Failure of core nodes*: We propose an autonomic mode where access nodes collaborate in order to offer the service even in case of failure of core service elements. This mode is transparent for end-users. Failure detection and restoration are dynamic and do not require any decision by human administrators.
- *Overload phenomenon*: Within this chapter, we describe a dynamic solution to prevent against flash crowds phenomena and prevent against the crash of service elements. Automatic setting of call acceptance ratio is enforced owing to a distributed decision-making process implemented by access nodes.

More details about the aforementioned solutions are provided in the remaining part of the chapter. Potentials and exploitation scenarios of autonomic networking paradigms into IMS-based architectures are also explained.

1.3 How this Chapter is Organized?

This chapter is composed of two main parts. The first part (Sect. 2) focuses on potentials of autonomic networking and distributed techniques into operational networks. This section discusses the motivations of the approach and especially the perception of end users (Sect. 2.1), the limits of business model of current deployed P2P service offerings (Sect. 2.2) and finally it identifies a set of viable exploitation scenarios (Sect. 2.4). The second part of this chapter (i.e. Sect. 3) focuses on solution space. Therefore three variants are described. Sect. 3.2 describes a solution for dynamic failure detection and restore targeting access nodes. Sect. 3.3 describes a dynamic failure detect and restore solution which is put into effect once core services nodes are unreachable or are out of service. Finally, Sect. 3.4 illustrates the usage of dynamic techniques to prevent and avoid flash crowds crisis.

2 Potentials of Autonomic Networking

This section aims to highlight the potentials of autonomic networking in Service Providers' service infrastructure. This section shows that, from the perspective of end-users (i.e., Sect. 2.1), architectural considerations are pointless. End-users are more careful about the availability of the services they subscribed to and to the level of perceived QoS and not how the service is (technically) built. Moreover, this section sketches the limits of the business model of some active actors in the field of P2P (see Sect. 2.2). These limits are provided as a warning to drive the decision-making of Service Providers. The interpretation of these limits within this chapter is to prevent other Service Providers to duplicate the technological choices of these active actors and to encourage adapting these techniques to their running environment.

This chapter focuses only on schemes targeting to adapt autonomic techniques within a centralized service platforms. The value creation considerations in pure autonomic realms are not detailed in this chapter and are out of scope of this chapter.

2.1 *The Perception of End-Users*

For end-users, or at least for the majority of them, the underlying technology used to deploy a given service offering is pointless. These users do not make any distinction between autonomic, P2P services (such as Skype), hybrid ones (e.g., Jajah) or centralized solutions (e.g., GoogleTalk, MSN, or Yahoo!). They only require a survivable service accessible at anytime and from anywhere, easy to use and to manage and (hopefully) free of charges. End-users subscribe to a given service platform where they can communicate with their contacts/friends leading to the emergence of isolated communities. Subscribing to a new service platform and leaving the former one is therefore not an easy task since all the community members have to migrate to the new service platform. This observation should not be anymore valid if service platforms are interconnected together and Service Providers go forward to abolish this new era of technology segregation. This leads us to conclude that sociological behaviors are more relevant and pertinent to consider rather than purely technological ones.

Another interesting aspect to consider is that end-users do not care about architectural considerations (e.g., the notion of hash tables, super-node, index) and are not familiar with the behaviors of the running P2P applications (e.g., lookup, routing, bandwidth consumption, memory load). For some P2P applications such as Skype, PPlive or Joost, end-users are not aware about the exploitation made by P2P services of their IP connectivity. Most of them are not aware that their resources are used by the P2P service even if they do not effectively use the service! For other P2P applications such as eMule, generally for file-sharing applications, end users are aware about the use of their Internet resources, because end-users understand that they need to share with external users so as the system lives.

2.2 The Business Limits of Some P2P Service Offerings

Skype was a success story. This success has been awarded by its affiliation to eBay in 2005 (eBay spent \$2.6 billion to buy Skype). This acquisition has been commented by the analysts and questions have been raised to evaluate how eBay will make benefit owing to this operation. These questions were left without credible answers until the first signals have been perceived (eBay wrote down its investment by \$900 million in October 2007). Recently, and as stated in [10], and despite its claimed 276 millions of registered users, 11.9 billions of Skype-to-Skype minutes and 1.6 billions of SkypeOut minutes during Q7 2007, and a cumulative 282 million dollars in 2007, eBay has overpaid Skype.

Former Skype CEO (and co-founder of Joost) has recently confessed this shared fact. These problems acknowledge the limits of the business model adopted by Skype. Indeed, Revenue sources of Skype are mainly the fees accounted by end users to use SkypeOut and SkypeIn services. According to the figures provided by eBay regarding its Q4 2007 financial exercise, SkypeOut revenue are stable (1.4 billions (min) during Q3 2007 compared to 1.6 billions during Q4 2007). Furthermore, the ratio of progress of new subscribers is 61% but the ratio of progress of SkypeOut minutes is only 10%. Note that eBay/Skype do not communicate about the interconnection costs and other charges, especially the software development ones.

The discussion above shows that the value creation is a problem for some P2P actors. Their business models should be rethought so as to reach a critical size to compete with Telcos.

2.3 A Service Provider Requirement: Toward Autonomic and Deterministic Networks

Evidently, Service Providers need to be convinced by the viability of the proposed autonomic schemes through concrete and realistic proposals which integrate Service Providers specificity and requirements. Particularly, incremental approaches should be investigated and proposed to Service Providers mainly because of the weight of backward compatibility.

Concretely, in order to promote autonomic networking and computing techniques within Service Providers community and to introduce these promising and emerging techniques within operational networks, Service Providers requirements should be taken into account. Indeed, autonomic networking designers should integrate in the proposed architectures the need of Service Providers to control the behavior of their deployed networks and built-on services. Service Providers should be able to assess the compliance of their offered/delivered services over the networks they are operating. They also need to check the level of quality of the services resulting from engineering operations, through measurements or other alternative means. Autonomic networking techniques should be designed in the context of end-to-end and cross-layer services.

The engineered networks must be compliant with the targets fixed by the service planning and design processes enforced by a given Service Provider. The behavior of deployed networks and offered services should be known in advance by Service Providers and no fuzzy behaviors should be experienced. In order to introduce autonomic techniques into real networks operated by Service Providers, operational constraints should be assessed. This requirement can be for instance implemented owing to elaborating torture tests and validation methodologies dedicated to a given autonomic techniques.

It is obvious that Service Providers need to go a step forward for enforcing service automation, dynamic provisioning and deterministic behaviors. This “ambition” is not motivated only by their need to reduce the OPEX but also for easing the service creation, service provisioning and maintenance, service troubleshooting, and also to reduce the TTM (time to market).

2.4 Autonomic Networking, a Promising Means to Enhance Service Availability and Robustness

Conversational services are one of the drivers for the emergence of enhanced IP architectures which should be QoS-aware, robust, and highly available. These features should not be implemented only at the IP layer but also at the service layer since the perceived service is cross-layer and cannot be exclusively considered as being the business of the transport layer. Moreover and as far as IP networks become the federative transport network, O&M-related functions become (more) critical. A single service can rely on the activation of several protocols and therefore constraints Service Providers to implement tools to ease the management of the services and the underlying protocol operations.

A long-term vision regarding the IP networking and associated services is that it should evolve toward a “Plug and Play” model which is characterized (partially) as follows:

- The introduction of new services is transparent for the service requestor and the network administrator. This is achieved through dynamic/automatic provisioning and auto-managed networks;
- Easiness is enabled everywhere:
 - Manageability of the proposed solutions
 - FCAPS (Fault, Configuration, Accounting, Performance, Security) functions
 - Service offer/negotiation/creation/execution/assurance/deletion
 - Service evolution/updating/maintenance
 - To estimate the interdependency of activated services

The above listed items are long-term objectives and cannot be implemented in one shot into operational networks in the near future. Indeed, an incremental approach should be adopted and migration scenarios investigated.

Since several proposals have been made in the field of service automation and dynamic provisioning [1, 5, 11, 19], we are in favor of investigating how autonomic means can be adapted so as to solve current lacks as encountered by service platforms such as IMS-based one. We are advocating for introducing distributed techniques and autonomic-like techniques to prevent against failures met by operational networks. We describe in Sect. 3, several scenarios and concrete solutions inspired from distributed techniques in order to enhance service robustness and availability of the offered services.

3 Solutions Space: Exploitation Scenarios of Autonomic Paradigm into IMS-Based Architectures

This section presents three proposals aiming to enhance the robustness of IMS-based architectures. These solutions are inspired from distributed and autonomic networking proposals. Each of the aforementioned solutions solves a failure case encountered by current IMS-based architectures. Indeed, Sect. 3.2 describes a lightweight solution avoiding physical redundancy of Session Border Controllers. (This redundancy is usually referred to as $2 * N$ in case of each SBC is backed up with a secondary SBC, and $N + 1$ when only an SBC is the backup of all primary SBCs.) Section 3.3 introduces a solution to ensure the service availability during the outage of core service elements or when a failure occurs between access nodes and core ones. The proposed solution is based on collaboration between access nodes so as to implement some of the functions usually enforced by core elements. Finally, Sect. 3.4 describes a way to prevent against overload phenomena mainly to prevent against flash crowds crisis. Note that the following section (i.e., Sect. 3.1) presents a tentative taxonomy and used terminology.

3.1 Taxonomy

IMS [4] and TISPAN (Telecoms & Internet converged Services & Protocols for Advanced Networks [20]) architectures have been specified by the 3GPP forum to meet the requirements of traditional Telcos, especially to allow the implementation of a convergent solution for both mobile and fixed service offerings. Some of big Service Providers have already started to deploy IMS-based solution mainly for their PSTN (public switched telephone network) renewal. It is not our intent to describe in depth the functional decomposition of those architectures. Only a high-level description of these architectures is presented hereafter. Indeed, an IMS-based service platform can be divided in several segments as illustrated in Fig. 1:

- *Access segment*: This segment groups functions which are required for connecting customers' equipment to the service. This segment may include, for instance, BGF (Border Gateway Function) or P-CSCF (Proxy Call Session Control Function). This segment is often represented as POP (point of presence). These POPs

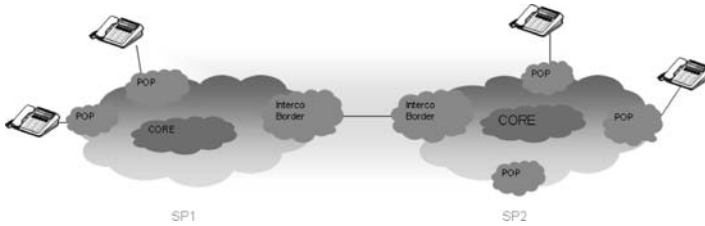


Fig. 1 VoIP Segments

are then connected to the core segment. For illustrating purposes, we assume that BGF function is embedded in a SBC (Session Border Controller [8]). One or several SBCs may be deployed per access POP. Each customer is provided/provisioned with its attached SBC which is the contact service element. SBC nodes implement several functions as listed in [8] such as topology hiding, fixing capability mismatch, and hosted NAT (Network Address Translator) traversal.

- *Core segment*: This segment is the “hearth” of the service. It is the place where the service logic and required functions such as routing and billing are hosted. Within IMS architecture, this segment is also responsible for interconnecting to internal or external AS (Application Servers). Examples of IMS functional elements which are located in the core segment are I-CSCF (Interrogating Call Session Control Function), S-CSCF (Service Call Session Control Function), HSS (Home Subscriber Server), etc.
- *Border segment or Interconnection segment*: This segment encloses all required functions to interconnect with adjacent service realms (including VoIP ones, PSTN, PLMN (public land mobile network), or any other voice service domain). This segment is critical since financial data depends on the records collected by this segment. In order to preserve confidentiality and to hide the internal service topology, embedded functions should support topology hiding and some signaling information such as SIP (Session Initiation Protocol [17]) headers must be modified or dropped. Examples of functional elements part of the border segment are SGF (Signaling Gateway), IBCF (Interconnection Border Control Function), and I-BGF (Interconnection Border Gateway Function).

Within this chapter, we will not describe in depth all functional elements defined in IMS/TISpan functional architectures. We focus only on a technical implementation of those architectures based on the deployment of SBC nodes at both access and border segments. We assume that for accessing to the service, customers are provisioned with appropriate information to reach their attached SBC. This former is the service contact point. The role of the SBC in this architecture is to relay received messages (both media and signaling) from user equipments (UEs) (also denoted as user agents (UAs)) to core service elements and vice versa. In the context of this chapter, we assume that both signaling and media messages are handled by the same SBC.

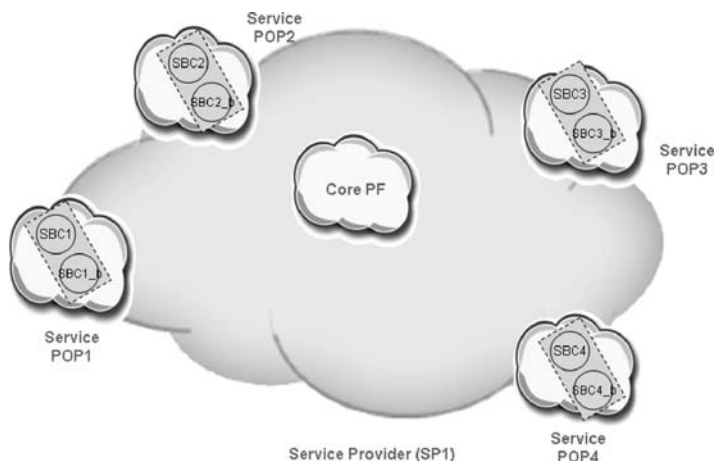


Fig. 2 Deployment Example of SBCs

Figure 2 provides an example of SBC deployment within the domain of a single Service Provider. This architecture is used as a reference architecture in the following sections.

Our reference architecture is composed of four service access POPs. Each customer (more specifically their terminals and service devices) is provisioned with appropriate information to reach one service access POP. In our example, each service access POP hosts an SBC pair (SBC_i, SBC_{i_b}) with SBC_i is the primary SBC and SBC_{i_b} is its backup that relays SBC_i when this latter is out of service. In addition to these POPs, the service administrative domain encloses also a cloud denoted by “Core PF” grouping all core service nodes. Only a macroscopic view of the core service segment is shown. In such architecture, the service is implemented through a coordinated cooperation between access service nodes and core ones. Access service nodes relay the message they receive from the UE toward the core service platform.

3.2 Failure of IMS Access Nodes

3.2.1 Scenario Description

This section treats the scenario of a failure encountered by access nodes. This outage may be due to a problem of an SBC or the unreachability of corresponding SBC for a set of end-users. Those end-users will not be able to access to the services they subscribed to. This scenario is illustrated in Fig. 3.

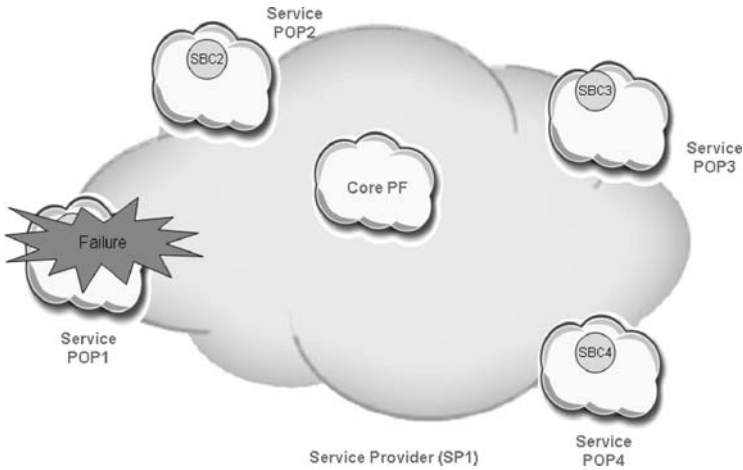


Fig. 3 Example of an Access Failure

3.2.2 How this is Solved in Nowadays Implementations?

In order to solve the problem described above, the current practices, as suggested by SBC vendors and implemented by some Service Providers, are mainly to dedicate another SBC as a backup to a primary SBC (see Fig. 2). At least two variants of this redundancy can be envisaged: (1) either to have a full redundancy of all primary SBCs or (2) to dedicate a set of SBCs to relay the ones out of service. The first variant, which is denoted as $2 * N$ model, requires an important number of nodes to be deployed by Service Providers. The second variant is an enhancement of the first one. Nevertheless, both variants are not optimal in terms of CAPEX and OPEX. Moreover, the current deployments suffer from a lack of “graceful” solutions for planned maintenance. Consequently, the service is still unavailable during the transition phase which is required for the backup SBC to be operational. Note that even the $2 * N$ model does not solve all failure scenarios such as an IP routing failure.

3.2.3 An Anycast-Based Solution

The main target of the solution introduced in this section is to ensure the service continuity and to avoid disruption when a given SBC encounters failures or when it is unreachable from customers’ sites. Concretely, this section introduces new means to maintain the same level of service as perceived by end-users. Unlike current state of the art solutions, our solution does not require any communication between a given SBC and its backup SBC. When failures are observed or detected, backup SBC replaces the primary one in a transparent manner to end-users and is dynamically enforced. Moreover, we rely on native IP techniques to solve this sensitive failure problem.

Within this solution, several IP addresses are assigned to the inner interface of SBCs (i.e., customer–SBC interface). These addresses are also assigned to other SBCs. Several modes of IP address assignment may be envisaged:

1. *Unilateral mode*: this mode assumes that a given SBC_i is the backup SBC of SBC_j . This latter is also the backup SBC of a third SBC_k with $SBC_i \neq SBC_k$. The inner interface is identified by two IP addresses, a primary and a secondary IP addresses. This secondary address is the primary address of another SBC. Note that distinct IP addresses are assigned to the outer interface (the one used to reach the core platform). Hereafter is provided an example of IP address assignment. The meaning of used notation is: the first element is the primary IP address. The second element is the secondary IP address. This address identifies the backup SBC. The third address identifies the inner interface IP address which is used to convey media traffic (RTP traffic). Finally, the fourth element identifies the IP address of the outer interface.
 - a. $SBC_i(@Ain, @Yin, @IPRTPA, @Aout)$
 - b. $SBC_j(@Bin, @Ain, @IPRTPB, @Bout)$
 - c. $SBC_k(@Cin, @Bin, @IPRTPC, @Cout)$
2. *Bilateral mode*: unlike the previous mode, SBCs are organized as pairs (SBC_i, SBC_j) with SBC_i is the backup SBC of SBC_j and vice versa. To implement this mode, the same IP address is assigned to the inner interfaces of SBC_i and SBC_j . The outer interface of each SBC is identified by distinct IP addresses. Below, we provide an example of IP address assignment. The first element identifies the primary IP address (which is assigned to two SBCs), the second element is the IP address used to convey RTP traffic in the inner interface (distinct IP addresses are used for this purpose), and finally, the third element which value is the IP address of the outer interface:
 - a. $SBC_i(@IPABin, @IPRTPA, @Aout)$
 - b. $SBC_j(@IPABin, @IPRTPB, @Bout)$
 - c. $SBC_k(@IPCDin, @IPRTPC, @Cout)$
3. *Distributed mode*: In this mode, SBCs are organized as groups. Each POP is composed by N customers attached to a given SBC. The Service Provider distributes its N customers to M groups. For each group a distinct IP address is provisioned. This address is the one to use to reach the primary SBC. For each SBC, several primary addresses are assigned to the inner interface. We provide hereafter an example of IP address distribution. In this example, $@IP1in$, $@IP2in$, and $@IP3in$ are the IP addresses assigned to the inner interfaces of SBC_i, SBC_j and SBC_k . This configuration allows to distribute the POP to three sub-groups. To each group, only one IP address is provisioned. $@IPRTPx$ is the IP address used to convey RTP traffic per SBC.
 - a. $SBC_i(@IP1in, @IP2in, @IP3in, @IPRTPA, @Aout)$
 - b. $SBC_j(@IP1in, @IP2in, @IP3in, @IPRTPB, @Bout)$
 - c. $SBC_k(@IP1in, @IP2in, @IP3in, @IPRTPC, @Cout)$
 - d. $SBC_p(@IP1in, @IP2in, @IP3in, @IPRTPD, @Dout)$

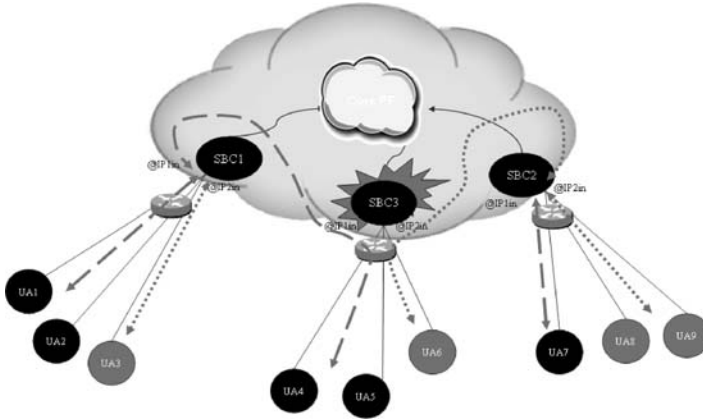


Fig. 4 Illustration Example of the Distributed Solution to solve Access Failures

Figure 4 shows an example illustrating the behavior of the IMS-based system when the configuration recommendations listed above are followed. In this example, we focus on the distributed mode. In this example, we suppose that the inner interface of SBC3 is configured with two IP addresses @IP1in and @IP2in. These two IP addresses are announced in IGP (Interior Gateway Protocol) by SBC3. UA4 and UA5, are provisioned to use @IP1in as the primary address to reach their contact SBC, and UA6 is provisioned with @IP2in. In the meantime, @IP1in and @IP2in are also assigned as the IP addresses if the inner interfaces of SBC1 and SBC2. Since these addresses are announced in the IGP, the SBC that handles the requests of UA4, UA5, and UA6 is SBC3 (because this is the shortest IGP path). In this section, we don't describe the configuration of IGP to enforce the proposed procedure). When SBC3 is out of service, the closest IGP path to reach @IP1in from UA4 and UA5 is the one toward SBC1. The one selected by IGP to reach @IP2in is the route toward SBC2. In this configuration, customers attached to SBC3 are dispatched between several SBCs in a transparent way.

Since the purpose of this section is to sketch a distributed method to handle the failure of access nodes, we will not provide elaborated specifications related to this solution. Based on the big lines provided above, we conclude that IP native solutions are feasible and viable to implement robust services without major investment and consumption of CAPEX/OPEX.

3.2.4 Migration Considerations

The aforementioned solution (see Sect. 3.2.3) is transparent to end-users and easy to implement into operational networks. Appropriate engineering guidelines and rules should be followed. New configuration data should be implemented and conveyed to corresponding nodes. Besides this provisioning operation, SBCs should be dimensioned accordingly. In order to avoid over-provisioning of all SBCs, we recommend the distributed mode described above. In such a model, all customers attached to a

given out-of-service POP will be dispatched between several SBCs. The proposed solution optimizes both CAPEX and OPEX and it allows intelligent distribution of customers.

3.3 Failure of IMS Core Nodes

3.3.1 Scenario Description

This section focuses on failures inducing a global service outage or partial one. In current deployments of conversational services and especially the ones based on IMS, an outage of the core service elements induces the unavailability of the service for all subscribed users. This outage may also be induced by an IP routing problem that occurs between access nodes and core ones. Consequently, the service can not be delivered to customers and therefore the offered SLA is not fulfilled.

Figure 5 shows a first scenario where only a subset of service access POPs are unable to relay service-specific messages to core nodes. Attached customers to those POPs will not access to the service. Figure 6 illustrates a second scenario with all POPs are unable to reach core service nodes. In such a case, all customers won't be able to use the service they subscribed to. This scenario could be due for instance to an electric outage of core service nodes. This scenario is even sensitive and the service "reboot" may take a long time before the service would be stable and all service elements behaviors would be "nominal." Furthermore, an avalanche restart will be experienced once all UEs detect that the service is available. An overload problem will be encountered by the service platform. This overload may be induced by the huge amount of registration messages that will be issued by UEs.

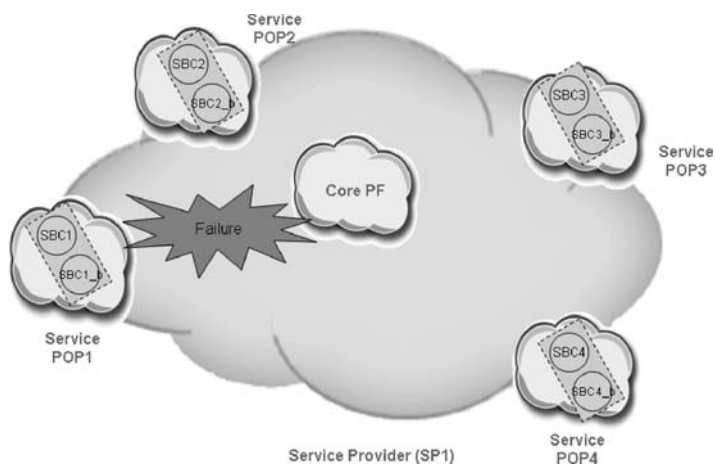


Fig. 5 Example of failures to join core service nodes

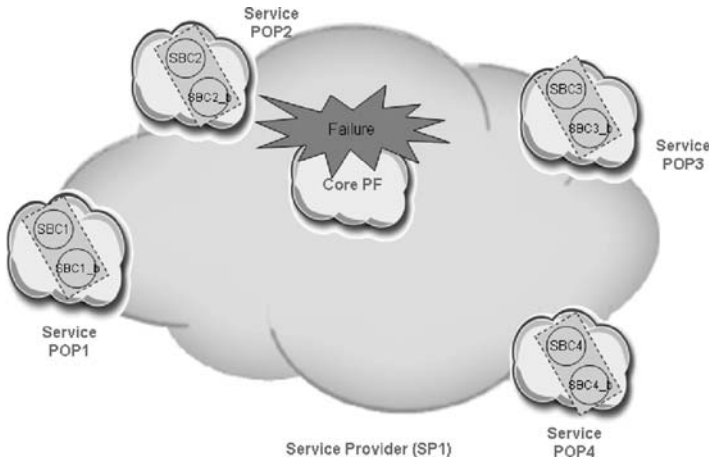


Fig. 6 Example of core nodes unavailability

3.3.2 How this is Solved in Nowadays Implementations?

In order to avoid the failure of core service nodes, current IMS-based Service Providers adopt a vertical integration of their service offerings. In such a configuration, the Service Provider owns also the IP infrastructure used to convey service-specific traffic. Appropriate planning dimensioning and engineering rules are enforced by the Service Provider. This facility is not suitable for “Pure” Service Providers which assumes a clear separation between the two business roles: IP Network Provider and Service Provider. For this category of Service Providers, layer-3- and layer-2-related failures may induce failure and therefore the un-reachability of IMS core nodes. In the next section, we propose an autonomous mode involving access SBC nodes which intervene in the service delivery chain. Failures of core nodes and problems to reach them will not be visible for end-users. Consequently, both perceived QoS and availability will be enhanced.

3.3.3 Distributed Mode

The delivery of a highly available service offering is a sensitive objective of Service Providers. Reaching this objective depends on the level of control Service Providers have on underlying transport (i.e., IP) infrastructure. Despite current deployment of IMS architectures assumes a vertical integration, IMS-based services can be offered without having a control on the underlying IP network. In this section, we introduce a solution to enhance the robustness of IMS-based architectures without assuming a vertically integrated system. Only high level description is provided. Detailed specifications are out of scope of this chapter.

The proposed solution advocates for involving SBC in the delivery of the services instead of “just” relaying messages from User Equipment to core service elements and vice versa. Concretely, SBCs assess the reachability of the core service platform

owing to a dedicated monitoring channel such as “keep-alive” messages which are sent to the core nodes. These “keep-alive” messages must be acknowledged. “Keep-alive” messages are exchanged periodically. Alternative “push” models may be implemented to assess the reachability of core service nodes. In the remaining part of this section, we do not detail the logic implemented to assess the reachability and the availability of the core service. We assume that appropriate procedures and algorithms are implemented in the core platform.

Once a given SBC detects that core nodes are out of service or no acknowledgments have been received, a specific message is sent to a multicast group [7] which members are all deployed SBCs. This message is denoted as **SBC_PROXY_REQUEST()**. The main function of this message is to check if at least one of the peer SBCs receives acknowledgments of its “keep-alive” messages. In such a case, the core service elements are still reachable from that SBC. Indeed, all SBCs which continue to receive acknowledgments to their “keep-alive” messages send an offer to the requesting SBC to become its **SBC_PROXY** owing to the invocation of a dedicated message referred to as **SBC_PROXY_OFFER()**.

- If several offers have been received by the requesting SBC, only one **SBC_PROXY** is selected. The role of an **SBC_PROXY** is to relay the messages received from a peer SBC to the core platform and vice-versa. The selection of an **SBC_PROXY** may be based on a set of criteria such as current load, CPU, and closest IGP path. As a result of this process, customers associated with a given SBC which encounters problems to reach core service elements (due to routing, link failures, etc), will be able to continue to access to their subscribed services owing to the presence of an **SBC_PROXY**. An example of a call flow is illustrated in Fig. 7.
- This scenario concerns only a partial failure (i.e., the example illustrated in Fig. 5). The next bullet point handle the scenario of global failure (all peer SBCs are unable to reach core service nodes, see Fig. 6).

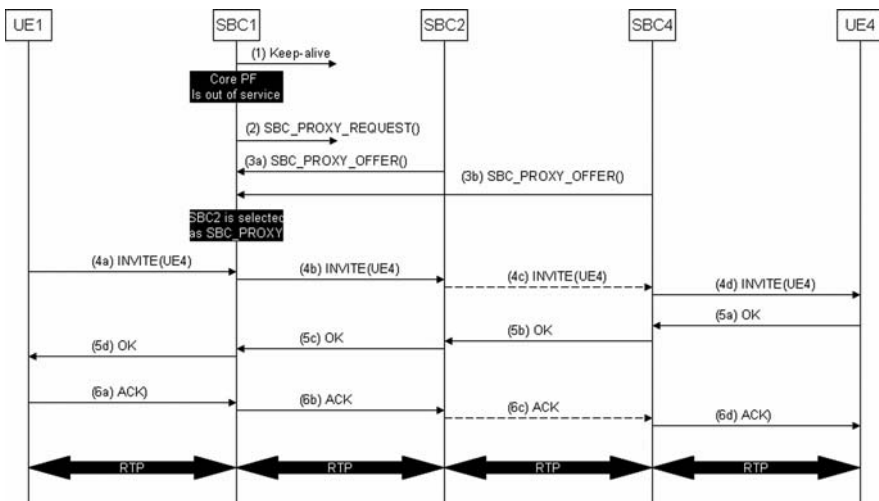


Fig. 7 Call flow with an SBC_PROXY in the path

- In order to place a call, three SBCs will intervene. To illustrate this, let consider the example illustrated in Fig. 7. In this example, UE1 is managed by SBC1 and UE4 is attached to SBC4. Suppose that a failure occurs and SBC1 is not able anymore to reach the core service nodes (*step 1*).
 - Since no acknowledgment messages have been received, a dedicated message, denoted as **SBC_PROXY_REQUEST()** message, is sent to the SBCs multicast group. This message is received by all peer SBCs (*step 2*).
 - In this example, SBC2 and SBC4 send an offer to SBC1 (*steps 3a and 3b*) by invoking **SBC_PROXY_OFFER()**.
 - In this example, SBC1 selects SBC2 as its **SBC_PROXY**.
 - When UE1 want to place a call toward UE4, it sends its **INVITE** message to its associated SBC (*step 4a*). This latter relays the request to SBC2 (*step 4b*) which in its turn relays the request to core service platform. Core nodes achieve routing operations and forward the request to SBC4. Finally the request is relayed to UE4. Once UE4 accepts this request, a dedicated message is routed via SBC2 (**SBC_PROXY** of SBC1) to reach UE1 (*steps 5a, 5b, 5c, and 5d*). The call between UE1 and UE4 can take place only after the delivery of **ACK** messages (*steps 6a, 6b, 6c, and 6d*). Then, media streams are exchanged between UE1 and UE4. These streams are not exchanged directly between endpoints (i.e., UE1 and UE4) but are relayed respectively by SBC1, SBC2, and SBC4. In current IMS deployments, this call would never take place since SBC1 is unable to reach core service nodes.
- If no **SBC_PROXY_OFFER()** message has been received after a dedicated timer expires, all SBCs adopt their autonomous mode and the messages are not relayed anymore to the core service nodes until these nodes became reachable again. The autonomous mode is described below.
 - All SBCs will actively contribute to the routing of the signaling messages.
 - To implement this mode, SBCs use their registration caching tables to resolve remote destinations as requested by end-users. The multicast channel quoted above is used by SBCs to resolve a given destination URI (Unified Resource Identifier).
 - Also are members of this multicast group, all interconnection nodes with adjacent telephony domains (such as PSTN and IMS). These nodes maintain a table grouping the list of (telephony) destinations reachable through them.
 - When the autonomous mode is adopted by all SBCs, the procedure to place a call is represented in the call flow shown in Fig. 8. Thus, when UE1 wants to reach UE4, it sends its session initiation request, via an **INVITE** message, to SBC1 (*step 3a*), after checking its registration cache table, a message to retrieve the contact address of UE4 is sent to the multicast group (*step 3b*). Since UE4 is managed by SBC4, this latter sends a response to SBC1 enclosing appropriate information to reach UE4 (*step 3c*). Then, SBC1 forward the **INVITE** message to SBC4, which in its turn forward it to UE4 (*steps 3d and 3e*). An acceptance message is then issued by UE4 and routed

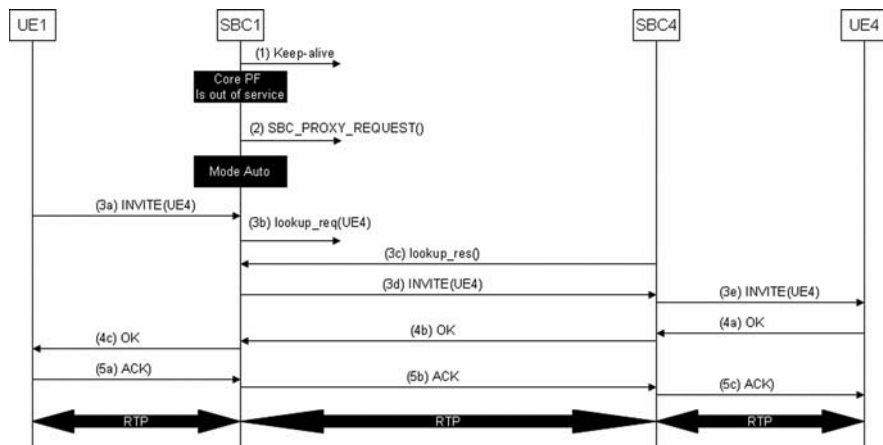


Fig. 8 Call flow: autonomous mode

back to UE1. Finally, media streams are exchanged between UE1 and UE4 via SBC1 and SBC4.

Owing to the activation of the solution above, failures are dynamically detected. The service is still being delivered to end-users thanks to the activation of the autonomous behavior of SBCs. Further investigations should be undertaken to check the feasibility to offer sophisticated services using this autonomous option.

3.3.4 Migration Considerations

The essence of the solution presented in Sect. 3.4.3 is to ensure the service availability in case of failures encountered by IMS core nodes or in case service access nodes fail to reach IMS core nodes. As stated above, these failure scenarios may impact the delivered services to all customers or to a subset of them. For the second scenario, a procedure to access to the service is put in place and a **SBC_PROXY** is selected. In such a case, no service degradation is to be observed by end-users. Moreover, the delivery of all subscribed functions would be possible. As for the first scenario, a basic service would be offered to end-users. The solution can be introduced in earlier stages of deployment to enhance the robustness of current and ongoing IMS-based service offerings and to avoid service unavailability as met in the current state of the art. In further steps, the challenge is to implement a full distributed service offering based on the solution proposed above. Concretely, the distributed mode should replace with no service degradation the centralized platform during failure events.

In the context of this chapter, we advocate for an incremental migration scenario. Indeed, we introduced this distributed mode as backup solution to avoid service unavailability and to ensure the service if still be delivered to subscribed customers. This positioning can be abolished once the solution is field proven. The solution presented in Sect. 3.4.3 can be promoted as primary mode to deliver the service. This target is a long-term objective and several technical hurdles should be solved before implementing this variant.

3.4 Flash Crowds Phenomena

3.4.1 Scenario Description

Several overload phenomena may be experienced by IMS-based architectures. As an example of overload crisis, this section focuses on the “flash crowds” problem. “Flash crowds” is an overload problem that occurs when a huge amount of traffic is issued by end-users to access to the service. This may occur during crisis events such as earthquakes or special events such as Christmas or during some TV shows. Due to this overload, the service offering may encounter some problems to handle all received requests and to deliver the service. This may even induce a crash of the service and its outage.

3.4.2 How this is Solved in Nowadays Implementations?

In order to solve the overload problem, SIP specifications [17] introduced a dedicated tag to notify users about the load status of the service. To do so, a SIP Proxy Server must issue a message with a code error equal to “503.” This message informs the users about the reason of rejection of their requests and asks them to send their request latter. A timer to re-issue the request is also enclosed in the error message: tag “retry-after.” The drawback of this approach is that the server must process an overhead load to notify all requesting end-users about its status. For more information about the drawbacks of this solution, refer to [16]. Another alternative to solve the overload problem is currently discussed within IETF. This solution is described in [14]. This latter document introduces a dedicated header denoted as “CONGESTION” to inform UEs about the load status of the server. Nevertheless, this solution suffers from the same drawbacks as the ones of “retry-after” tag. [9] proposes an elaborated method to indicate to the requesting elements to reduce the rhythm of sending their requests. A new SIP header is introduced for this purpose: “Load Header.”

3.4.3 Distributed Mode

The principle of this solution is to assume that the load of the core service platform may be deduced from the one injected by each SBC toward the core service platform. Let suppose that N is the overall load of the core service platform and M_i is the load of a given SBC $_i$. N can be computed from M_i of each individual SBC $_i$ as follows: $N = f(M_i)$ with $1 \leq i \leq p$ and

- $f(M_i) = \sum M_i, 1 \leq i \leq p$, if only one and only one SBC intervenes in the placement of a call.
- $f(M_i) = \frac{1}{2} * \sum M_i, 1 \leq i \leq p$, if two SBCs intervene in the placement of a call.
- $f(M_i) = \frac{1}{k} * \sum M_i, 1 \leq i \leq p$, if k SBCs intervene in the placement of a call.

This formula shows that the overall traffic may be tuned and controlled by setting appropriately the acceptance ratio of each SBC $_i$, i.e., to set a threshold m_i per SBC $_i$

so as $f(Mi) < n * N, 1 \leq i \leq p$, with n is the authorized overload ratio. We propose to introduce a new function responsible for setting individual mi per SBC. This function may be supported by a centralized element, which is not necessarily part of the core service platform. It may also be enforced by SBCs themselves. In this chapter, we describe only this latter scheme, which is denoted as *distributed mode*. Two variants of the distributed mode may be envisaged:

- Fully autonomous mode*: In order to implement this mode, a point-to-multipoint (P2MP) channel must be configured. This channel groups all SBCs deployed in the platform. Each SBC sends periodically or based on some internal/external events its current load (Mi) and its acceptance ratio (mi) owing to the invocation of an **RPT** message. This notification is received by all peer SBCs member of the P2MP channel (see Fig. 9). Once received, remote SBCs update their service load table and compute the overall load as should be experienced by core service node owing to invocation of the formula defined above. If the overall load is close to $n*N$, then a decision-making process is enforced by each SBC to determine a new threshold mi . Several logic may be adopted for the decision-making process such as the SBC which injects big amount of traffic should decrease its acceptance ratio or all SBCs acceptance ratios should be reduced by $x\%$. This mode is complex to implement and the consistency of the overall system behavior is difficult to assess. For these reasons, we introduce a second variant called **MASTER_SBC** mode.
- Mode with MASTER_SBC*: This mode introduces a novel role, denoted as **MASTER_SBC**, assigned to an SBC. Instead of having a distributed decision-making algorithm, only one SBC implements it and sends appropriate decisions to remote SBCs. The assignment of the SBC responsible of implementing **MASTER_SBC**-related functions may be static (i.e., configured by the Service Provider) or

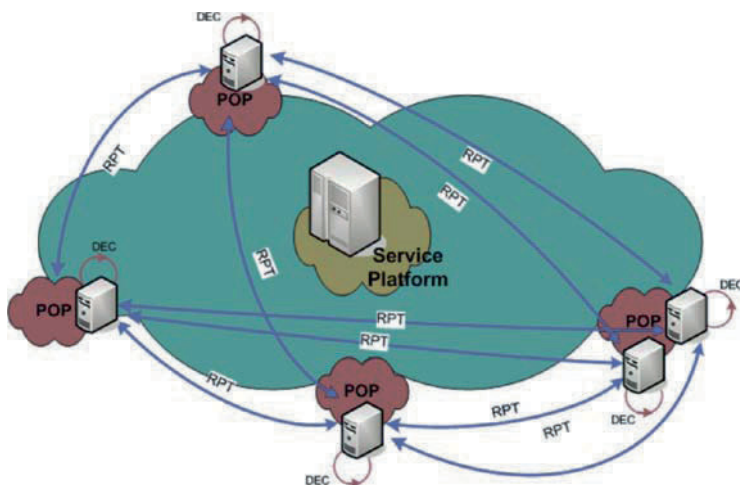


Fig. 9 Distributed mode to solve overload problem

dynamically elected through appropriate means which are out of scope of this chapter. In the remaining part, we assume that an SBC acts as a **MASTER_SBC**. Like the distributed mode, all SBCs send their load status and their current acceptance ratio to a P2MP group. This notification is received particularly by the **MASTER_SBC**, this latter computes the overall load as should be observed by the core service platform. If the overall load is close to $n * N$, a decision is then issued and sent to a subset of SBC so as to have the overall load less than $n * N$. Note also that the ratio mi may be increased for a given SBC when no risk to induce an overload is foreseen. This mode is dynamic and does not require any intervention of core service nodes. Moreover, the system can be adapted so as to automatically tune individual acceptance ratios to prevent against overload. Doing so, the Service Provider ensure the stability of its offered services.

3.4.4 Migration Considerations

Since the proposed solution focuses on the border elements, notably SBCs, the activation of the proposed procedure (Sect. 3.4.3) can be envisaged in operational service platforms. In a first step, current backup SBCs can be updated to support required behaviors as described in Sect. 3.4.3. Once this is achieved, a planned maintenance operation should be scheduled. Backup SBCs will then replace primary nodes in the service delivery chain. The next step is then to update the primary SBCs and reschedule another planned maintenance operation so as to allow primary nodes to intervene in the service delivery chain. It is recommended to implement “graceful” means for planned maintenance operations. Concretely, no service disruption should be observed by end-users when backup SBCs replace the primary ones. Replacement of primary SBCs should not be in one shot. A procedure to avoid service instability should be privileged. When all nodes are updated with required functions to implement the solution described in Sect. 3.4.3, SBCs will intervene to dynamically adjust their acceptance ratios in such a way IMS core nodes would not be overloaded.

4 Conclusions and Future Trends

This chapter described viable tracks for enhancing current centralized IMS-based solutions owing to the implementation of techniques inspired from autonomic networking and more especially from distributed systems. Several robustness vulnerabilities encountered by nowadays IMS-based architectures have been identified and solutions provided. Moreover, migration issues are also taken into account. This chapter has introduced concrete exploitation and implementation scenarios of autonomic networking into operational networks.

This chapter has highlighted the need of more elaborated research activities so as to meet the requirements of Service Providers mainly to ease deployment of deterministic systems. Indeed, Service Providers must be able to “know” in advance the

behavior of the systems and solutions they deploy. It is not acceptable from a Service Provider perspective, and more especially for Telcos, to enable solutions which outputs are “fuzzy.” This requirement is critical since service offerings are usually associated with service level agreements (SLAs) which must be fulfilled by Service Providers. To enforce this constraint, Service Providers should implement service assurance functions so as to check the level of the offered service as perceived by end-users. Service offerings of the future should be designed with hard requirements on service assurance functions. Appropriate interfaces, put at disposal of end users in order to check the conformance of provided services with regards the SLA they subscribed to, should be activated.

Introducing autonomic networking techniques inside operational networks should be incremental. Techniques for dynamic provisioning and service automation should be promoted within Service Providers community. Besides this concern, self-care techniques and service sanity checking means should be proposed so as to meet the requirements of Service Providers. The aim of this chapter was to contribute to the dissemination of autonomic networking from the standpoint of Service Providers.

References

1. J. Boyle et al., “The COPS Protocol”, RFC 2748, IETF, January 2000
2. M. Boucadair et al., “Challenges of Next Generation Conversational Services: Quality of Service and Migration to IPv6”, Proc. of the 8th GRES’07 Colloquium, Tunisia, November 2007.
3. M. Boucadair, Y. Noisette, “Migrating SIP-Based Conversational Services to IPv6: Complications and Interworking with IPv4”, 2nd ICDT, USA, July, 2007.
4. G. Camarillo and M. A. Garcia-Martin, “The 3G IP Multimedia Subsystem- merging the Internet and the cellular worlds”, John Wiley, 2005
5. Chan, K. H. et al., “COPS Usage for Policy Provisioning”, RFC 3084, March 2001.
6. Deering S., Hinden R., “Internet Protocol, Version 6 Specification”, RFC 2460, December 1998
7. Deering, S., “Host Extensions for IP Multicasting”, RFC 1112, August 1989
8. J. Hautakorpi et al., “Requirements from SIP Session Border Control Deployments”, draft-ietf-sipping-sbc-funcs, March 2008
9. Hilt, V. et al., “Sessions Initiation Protocol (SIP) Overload Control”, work in progress, March 2007
10. “Voice-Over-Internet phoning doesn’t make sense’ in market”, International Herald Tribune, 11 February 2008
11. Jacquenet, C., Boucadair, M., Bourdon, G., “Service Automation and Dynamic Provisioning Techniques in IP/MPLS Environments”, John Wiley & Sons Inc, March 2008
12. Yensy, J.H. et al., “Joost: A Measurement Study”, May 2007, available at <http://www.patrickpiemonte.com/15744-Joost.pdf>
13. Maccarthaigh, C., “Joost Network Architecture”, RIPE, April 2007
14. Malas, D., Terpstra, R., “The Session Initiation Protocol CONGESTION Header Field”, work in progress, May 2006
15. Melcher, B, Mitchell, B. “Towards an Autonomic Framework: Self-Configuring Network Services and Developing Autonomic Applications”, Intel Technology Journal, 2004
16. Rosenberg, J., “Requirements for Management of Overload in the Session Initiation Protocol”, work in progress, November 2006

17. Rosenberg, J. et al., "SIP: Session Initiation Protocol", RFC 3261, June 2002
18. Rosenberg, J., Squire, M., and H. Salama, "Telephony Routing over IP", RFC 3219, August 2001.
19. S. Sengupta and R. Ramamurthy, "From Network Design to Dynamic Provisioning and Restoration in Optical Cross-Connect Mesh Networks: An Architectural and Algorithmic Overview". IEEE Network Magazine, Vol. 15, No. 4, July/August 2001.
20. TISPAN, "Telecommunications and Internet converged Services and Protocols for Advanced Networking, NGN Release 1", TR180001, 2006
21. Postel, J., "Internet Protocol", RFC 791, September 1981
22. Jun, L. et al., "An Experimental Analysis of Joost Peer-to-Peer VoD Service", October 2007, available at http://www.net.informatik.uni-goettingen.de/publications/1484/Joost_experiment.pdf

Embodied Cognition-Based Distributed Spectrum Sensing for Autonomic Wireless Systems

Luca Bixio, Andrea F. Cattoni, Carlo S. Regazzoni, and Pramod K. Varshney

Abstract In the past decade, the usage of portable communication devices has continued to increase. Autonomic communications (AC) represents a new frontier for mobile communications because they will allow autonomous and self-regulated network and communication protocols procedures. Dynamic observation of the spectrum and adaptive reactions of the autonomic terminal to wireless channel conditions are hence important problems in improving the spectrum efficiency as well as in allowing a complete access to the network wherever and whenever the user needs them. Cognitive radio probably represents the most suitable paradigm for building communication terminals/devices for AC. In this chapter, after a tutorial overview of the current state of the art on cognitive radio visions and on stand-alone and cooperative/distributed approaches to spectrum sensing, the general problem of spectrum sensing will be addressed. Then a new vision, based on embodied cognition will be presented together with a distributed spectrum sensing algorithm that is formalized within the embodied framework. Results will illustrate the effectiveness of the proposed method.

1 Cognitive Radio for Autonomic Wireless Communications

In the past few years, the usage of portable communication devices has continued to increase at a rapid pace [3]. Together with mobile devices, new communication services have been proposed thanks also to new communications standards. On the one hand, such new standards provide flexibility in communications to end users. But on the other hand, this places a huge demand for radio spectrum that is expected to grow in the future [37]. To allow for such rapid growth, different frequency bands in the radio spectrum are selected and assigned to different standards by

L. Bixio (✉)

Department of Biophysical and Electronic Engineering, University of Genova, Via Opera Pia 11a, 16145 Genova, Italy
e-mail: luca.bixio@dibe.unige.it

governmental regulatory agencies [23] in order to guarantee coexistence between different services [37].

After many years of fixed radio spectrum assignment in order to meet the increasing demands due to emerging services, the unlicensed frequencies are going to disappear [17]. In fact, a study conducted by the US Federal Communication Commission (FCC) [15] has pointed out that the radio spectrum is heavily crowded with most frequency bands already assigned to licensed users for a given service [37]. Moreover, the variation in estimated use of licensed spectrum ranges from 15% to 85% [1], while the Defense Advance Research Projects Agency (DARPA) [26] estimated that only 2% of the allocated spectrum is in use in the USA at any given moment. For these reasons, it is clear that a flexible utilization of the radio spectrum is necessary. In fact, according to Haykin [23], “in many bands, spectrum access is a more significant problem than physical scarcity of spectrum, in large part due to legacy command-and-control regulation that limits the ability of potential spectrum users to obtain such access.”

This means that radio spectrum utilization can be significantly improved if unlicensed (secondary) users are allowed to access licensed bands if and only if at a given time and in a given location licensed (primary) users are not using it. It has now become abundantly clear that a dynamic management of radio spectrum allocation is required to meet the growing demand and to have efficient utilization of the spectrum. To facilitate this, continuous and dynamic observation of the radio spectrum in order to adaptively react to wireless channel conditions are important issues in improving radio spectrum utilization [23]. In a broad sense, the cognitive radio provides different solutions in order to solve some of these problems [16]. A broad survey of cognitive radio approaches will be provided in Sect. 2.

Haykin [23] provides the following definition for cognitive radio:

Cognitive Radio is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and adapt its internal states to statistical variations in the incoming RF stimuli by making corresponding changes in certain operating parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in mind:

- highly reliable communications whenever and wherever needed;
- efficient utilization of the radio spectrum.

As it is clear from this definition, the common keywords for an efficient cognitive radio are *awareness* and *reconfigurability*. In a radio environment, awareness means the capability of the cognitive radio to understand, learn, and predict what is happening in the radio spectrum [16], that is, cognitive radio is able to identify

the transmitted waveform, to localize the radio sources, etc. Spectrum awareness is also known as spectrum sensing and it will be addressed in detail in Sect. 3. Reconfigurability is necessary to provide *self-configuration* of some internal parameters according to the observed radio spectrum [23]. Reconfigurability provides *self-optimization* [25] of the cognitive radio in order to accommodate new standards and new services as they emerge [16]. Furthermore, reconfigurability is enormously important for both civilian and military applications especially when unforeseen situations happen and some network infrastructures are not available providing *self-healing* and *self-protection* capabilities.

The capabilities listed above perfectly match with the autonomic computing vision proposed in [25]. In Table 1 a *self-management* capabilities comparison among classical autonomic computing system [25] and cognitive radio system is provided.

The first example of a cognitive radio system equipped with the capabilities listed in Table 1 has been considered in the DARPA NeXt Generation (XG) radio development program [13, 26]. This cognitive radio senses the radio environment, identifies an opportunity in which secondary users can transmit in a given licensed frequency band, adapts the transmission parameters in order to exploit the detected opportunity, transmits, and releases the occupied band if a licensed user accesses it [16]. These tasks are cyclically executed according to stored experience provided by a learning process.

Table 1 Self-management capabilities comparison among autonomic computing system and cognitive radio system

Ability	Autonomic computing	Cognitive radio
Self-configuration	Automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly [25].	Automated optimal configuration of transmission parameters according to spectrum sensing in order to avoid harmful interference to licensed users.
Self-optimization	Components and systems continually seek opportunities to improve their own performance and efficiency [25].	Systems continuously perform spectrum sensing to detect opportunities to improve their own performance and spectrum utilization.
Self-healing	System automatically detects, diagnoses, and repairs localized software and hardware problems [25].	System automatically vacates the occupied band if a licensed user attempts to access it.
Self-protection	System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent systemwide failures [25].	System automatically defends against malicious attacks and avoids cascading failures while detecting opportunities.

2 Cognitive Radio Approaches

2.1 Mitola's Definition

One of the main contributors to the definition of the cognitive radio paradigm was Joseph Mitola III. In [29], he defined the cognitive radio as a system that “can track the user’s environment over time and space. Cognitive radio, then, matches its internal models to external observations to understand what it means to commute to and from work, take a business trip to Europe, go on vacation, and so on.” From this definition, the intrinsic capabilities of autonomy, transparency, and learning are evident. In his vision, such capabilities have to be implemented using a common META-Language (MTL) that he defines as Radio Knowledge Representation Language (RKRL). It is useful to describe, at a semantic level, according to classical artificial intelligence (AI) vision, “space-time models of the user, network, radio resources, and services” that can “personalize and enhance the consumer’s experience.” He models the behavior of the proposed cognitive radio using a state/transition representation known as a *cognitive cycle* (CC) [29]. It represents the possible time-varying states the cognitive radio can assume and which are the transitions that link the states with each other. In Mitola’s CC, the transitions are triggering events or situations that can happen in the surrounding environment (e.g., external world).

Mitola focused his work on the mass market civilian applications. As a matter of fact, he was interested in the impacts of a dynamic and multipurpose cognitive device on the possible service provisioning from a network provider side. Starting from this civilian- and market-oriented framework, this original work (and the following ones) gave a great impetus to the scientific community for facing the various open issues/challenges in order to implement a practical, cognitive radio device, with self-management capabilities.

2.2 Haykin's Definition

Six years after Mitola, another important researcher, Simon Haykin, provided a more precise and detailed definition of cognitive radio in his paper *Cognitive Radio: Brain-Empowered Wireless Communications* [23]. As stated in Sect. 1, in autonomic wireless communications, a cognitive radio can be defined as a system provided with some sort of *intelligence*. Such a system is able to sense the surrounding environment and using a “methodology of understanding-by-building to learn from the environment” adapts its internal parameters in order to achieve two global goals: (1) “highly reliable communications” and (2) “efficient utilization of the radio spectrum” [23].

While Mitola was mainly interested in the impact of the cognitive and self-management capabilities onto the communications market. Haykin addressed the problem from a more general point of view providing a more detailed and explicit definition. Both agreed on the fact that the Software-Defined Radio (SDR) systems [29] can be used for developing a practical and efficient cognitive radio. Furthermore, according to Haykin [23], “Software-Defined Radio (SDR) is a practical

reality today, thanks to the convergence of two key technologies: digital radio, and computer software.” It is now clear that in order to implement a cognitive radio, it is necessary to provide some cognition capabilities (sometimes also known as *intelligence* or *smartness*) to a flexible and highly reconfigurable system, provided by the SDR architecture [23].

The behavior of the cognitive dynamic system proposed by Haykin can be represented by a CC [23], similar to Mitola’s one [29] (as explained in Sect. 2.1), but much more clustered in macroprocesses. This behavioral approach is based on three macrostates which establish, the cognitive foundations of the cognitive radio [23]:

- “Radio-scene analysis”: Cognitive radio has to detect opportunities and adapt its transmission parameters in order to avoid harmful interference to primary users.
- “Channel identification”: Cognitive radio has to estimate the channel state information (CSI) in order to predict the channel capacity that it can exploit.
- “Transmit-power control and dynamic spectrum management”: Cognitive radio has to perform power control and dynamic spectrum utilization in order to achieve the above listed tasks.

Such active states and their characteristics define the main characteristics of a cognitive radio from a signal processing/communications point of view and they are widely accepted by the cognitive radio scientific community.

2.3 Other Visions

In [22], Palicot discusses his idea for the evolution of a cognitive radio device from a common SDR platform. In fact, he proposes to add self-management capabilities in order to provide awareness to a reconfigurable radio and he supplies some assumptions for adding such capabilities to the considered system [22]:

Sensing means refer to all the possible methods the cognitive radio system has at its disposal for observing its environment, which can be categorized in four main families described below:

- *electromagnetic environment*: spectrum occupancy, signal-to-noise ratio (SNR), multipath propagation, etc.
- *hardware environment*: battery level, power consumption, computational resources load, etc.
- *network environment*: telecommunication standards (GSM, UMTS, WiFi, etc.), operators and services available in the vicinity, traffic load on a link, etc.
- *user-related environment*: position, speed, time of day, user preferences, user profile (access rights, contract, . . .), video and audio sensor (presence detection, voice recognition), etc.

Such collected observations are stored and processed to provide self-awareness to the cognitive radio system. Palicot's work is focused on the design of a cognitive "engine" for implementing the reconfiguration operations on the SDR platform.

Other researchers are much more interested in the self-management and learning capabilities of the cognitive radio, such as Doyle and Sutton. In [31] and [39], they design the high-level cognitive capabilities of their platform through classical AI approaches and the usage of a META-Language. In their system, both the CC and the reconfiguration manager, a type of middleware able to control hardware and software reconfiguration abilities, has been implemented by programming languages such as the eXtensible Markup Language or the Web Ontology Language.

Another semantic rule-based approach is the one proposed by Clancy et al. in [10]: "a cognitive radio extends a software radio by adding an independent cognitive engine, composed of a knowledge base, reasoning engine, and learning engine, to drive software modifications." All these characteristics are implemented in semantic or subsemantic states which interact with each other through binary logic operators. Clancy's cognitive radio is implemented on an open source Software Communications Architecture and it is able to learn from the acquired knowledge.

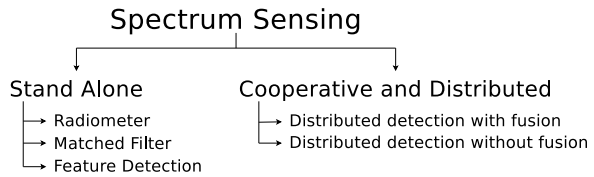
In [16], Bostian et al. propose a system that jointly exploit a feature based optimization algorithm and a classical case-based reasoning engine. The strength of the system relies on the multiobjective decision making (MODM) approach for optimal reconfiguration. It takes into account all the different quantitative parameters about the goodness of the wireless link, such as packet delay, data rate, SNR, the fading statistics, and it jointly optimizes all the parameters of the Physical (PHY) and Link (LLC) levels of the ISO-OSI stack. The optimization is performed using a genetic algorithm which allows us to find optimal solutions in multidimensional and heterogeneous optimization spaces.

3 Spectrum Sensing

As has been pointed out in the previous sections, a cognitive radio has to be able to sense the environment over a wide portion of the spectrum and autonomously adapt to it since the cognitive radio does not have rights to any frequency bands.

This task performed by cognitive radio is known as spectrum sensing [1, 16, 23] (or Spectrum Monitoring [17–19]). Generally speaking, spectrum sensing in wireless communications is one of the most challenging tasks that a cognitive radio has to perform. Depending on the required level of automation and self-management capabilities, spectrum sensing has to provide to the cognitive radio different information in order to predict the radio spectrum utilization. For these reasons, in some applications, providing information only about the frequency usage would not be sufficient, and other characteristics about the portion of the spectrum under investigation have to be provided in order to predict the radio spectrum utilization (e.g., number of transmitted signals, carrier frequency, power, transmission technique, modulation). In fact, prior knowledge about the transmitted signal and

Fig. 1 Classification of spectrum sensing techniques: stand alone, and cooperative and distributed



its parameters (e.g., carrier frequency, power, modulation) is usually not available. Moreover, received signals are corrupted by channel distortions (e.g., severe multipath fading), and spread spectrum transmission techniques are often used in order to obtain a low probability of interception.

Generally, spectrum sensing techniques can be classified as stand alone, and cooperative and distributed, as shown in Fig. 1.

In the following sections a survey on these spectrum sensing techniques will be provided and advantages/disadvantage for the different approaches will be discussed.

3.1 Stand-Alone Spectrum Sensing

Stand-alone spectrum sensing techniques have been treated extensively in the literature [1, 2, 8, 14, 16, 17, 20, 21, 34, 41]. This kind of techniques have been studied for military and civilian applications for signal detection [21], automatic modulation classification [14], radio source localization [8], and communication jamming [21] purposes.

In the past, the most commonly used approach to spectrum sensing was based on *energy detector* [21, 41] (or *radiometer*), that is measurement of received energy in selected time and frequency intervals. Radiometer is one of the most used techniques thanks to its low computational load. However, it is well known that this strategy is highly sensitive to unknown and varying noise level [21]. In order to overcome this limitation, some modified radiometers, with adaptive thresholds and filtering, have been proposed [20]. In spite of these modified approaches, unknown and varying noise level is the most serious impediment to reliable spectrum sensing [21]. Moreover, in the past decade, different spread spectrum transmission techniques have been proposed in order to obtain a low probability of interception. If such techniques are used, received signal power is close to the noise threshold (or sometimes under, i.e., negative SNR) and it is undetectable by a radiometer without increasing the false alarm probability [21].

When some information about the transmitted signal is known to the cognitive radio, the optimal detector, under assumption of stationary Gaussian noise, is the *matched filter* since it maximizes the received SNR [34]. The matched filter requires perfect knowledge of the transmitted signal parameters, such as modulation type, order, and pulse shape in order to provide optimal detection. But if perfect knowledge of the transmitted signal is not available or it is not accurate, the performance

of the matched filter degrade quickly [1]. In a cognitive radio environment a priori knowledge about the transmitted signal is usually not available. In spite of this, it is possible to use a matched filter in a cognitive radio that relies on SDR [23] as it is able to autonomously select the correct filter according to the radio environment under investigation. This means that a wide range of matched filters (one for each signal that is expected to be present in the considered radio environment) have to be implemented on a software platform with self-management capabilities. Thanks to self-management and reconfigurability capabilities, it is still possible to obtain optimal detection at the price of high computational load. For this reason, the matched filter approach is not suitable for practical cognitive radios, especially when a crowded frequency band is considered (e.g. Industrial, Scientific, and Medical band).

An alternative spectrum sensing technique is based on *feature detection*. In this context, a feature is defined as an inherent characteristic which is unique for each class of signals. In the literature [1, 2, 14, 16, 17, 21, 32, 40], different features have been considered in order to detect and classify signals in a given radio environment. Some of the most intuitive features considered are instantaneous amplitude, phase, and frequency [2]. Such features are usually used to detect and classify linear modulation [14].

More recently, analog-to-digital conversion has made the use of transforms practical [16] in order to localize the changes in instantaneous amplitude, phase, and frequency. Typical transforms used are discrete fourier transform [16], wavelet transform [14], and Wigner–Ville transform [17]. The above-mentioned feature detection approaches have advantages depending on the considered application, computational complexity, and radio environment.

One of the most used and interesting feature detection technique is based on the cyclic feature. This technique was first introduced by Gardner in [21] for signal interception purposes but, in the previous years, Jondral et al. [32] and Doyle et al. [40] have proposed the use of the cyclic feature as a spectrum sensing technique for cognitive radio applications. Cyclic-feature detection approaches are based on the fact that modulated signal are usually coupled with sine wave carriers, hopping sequences, cyclic prefixes, spreading codes, or pulse trains, which result in a built-in periodicity [1]. These modulated signals are said to be cyclostationary since their mean and autocorrelation functions exhibit periodicity [21]. Such periodicity can be used as a feature and can be detected by analyzing a spectral correlation function (SCF) [1, 21], also known as cyclic spectrum [21]. The main advantage obtained by using SCF analysis is that it is possible to distinguish between noise and signal (even at negative SNR) thanks to the fact that noise is a wide-sense stationary random process [33], with no spectral correlation, while the modulated signals are cyclostationary, with spectral correlation due to embedded periodicity. Therefore, a cyclic-feature detector can overcome the energy detector limits in detecting signals in low SNR environments [1]. In fact, signals with overlapping features in the power spectrum, can have nonoverlapping features in the cyclic spectrum [21]. Moreover, the cyclic spectrum is a much richer domain for signal detection than classical power spectrum. This property allows us to use this technique as a more complete tool [21]

for spectrum sensing. In spite of these advantages, cyclic-feature detection is computationally complex and requires significantly long observation time [1].

3.2 Cooperative/Distributed Spectrum Sensing

In spite of the advances made on stand-alone techniques, spectrum sensing can remain a complex task when “difficult” scenarios are considered [16]. In real radio environments, the received signal is corrupted by multipath fading, frequency selectivity, time varying channels, and noise [23]. In fact, it is well known that radio propagation across a wireless channel is affected by path loss (that is, received signal power decreases with the distance between transmitter and receiver) and shadowing (that is, received signal power fluctuates around the path loss) [23]. These phenomena can cause significant fluctuations of the signal level at the cognitive radio, which is then unable to perform reliable spectrum sensing [23] if a stand alone technique is used. This is of particular importance in cognitive radio, since a “false opportunity” could be detected due to a sudden fading of the received signal caused by multipath resulting in incorrect spectrum allocation.

In order to overcome such limits and to improve the performance of spectrum sensing, cooperative and distributed techniques have been proposed [12] in order to exploit spatial diversity [36] inherent in cognitive radios that are geographically separated in the considered environment. Such techniques may significantly improve the reliability of spectrum sensing at the cost of increased computational complexity and bandwidth usage for exchanging information among cognitive radios [12]. It is necessary to remark that additional algorithms are needed in order to combine shared information about “local” spectrum sensing. Moreover, a dedicated feedback channel has to be allocated in order to share collected information. When cognitive radio applications are considered where a dedicated channel is not available [23], other methods which require low or no overhead should be considered [23].

To this end, different methods are available in the literature [42] and they can be categorized on the basis of the exchanged information between cognitive radios [42] (or on the basis of the computational capabilities of the cognitive radio). According to Varshney’s distributed detection theory book [42], it is possible to identify two classes of distributed spectrum sensing techniques: *distributed detection with fusion* and *distributed detection without fusion*.

In the former [12], a set of N cooperative cognitive radios share the same radio environment. Each cognitive radio performs spectrum sensing by one of the techniques proposed in Sect. 3.1 according to its computational capability. Then, it sends the output of the spectrum sensing task to a data fusion center, which provides a “global” spectrum sensing decision based on gathered data [12]. It is necessary to remark that in this context, different solutions can be proposed depending on the level of cooperation among cognitive radios [42]. An example of distributed detection with fusion is shown in Fig. 2. Although distributed detection with fusion spectrum sensing techniques achieve better performance than stand-alone spectrum

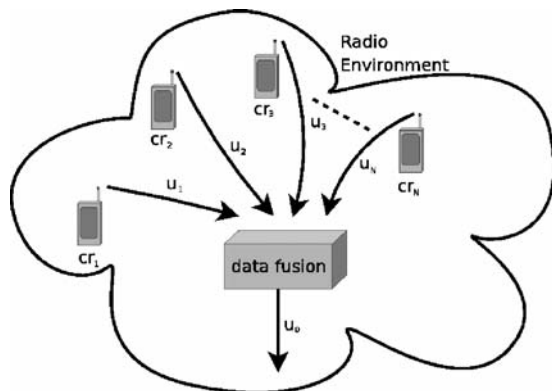


Fig. 2 Example of distributed detection with fusion scenario: N cognitive radios share the same radio environment and observe it. Each cognitive radio performs spectrum sensing using one of the proposed techniques in Sect. 3.1. It sends the output of its processing ($u_i, i = 1, \dots, N$) to a data fusion center. This center computes a global spectrum sensing decision u_0 based on received messages u_i

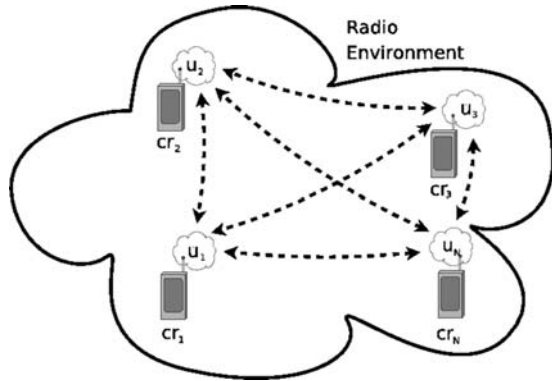
sensing, there are some open issues for implementing them in practical cognitive radio applications [42]:

- a dedicated channel to share observations may not be available;
- on the one hand, a dedicated channel can improve the performance of spectrum sensing; but on the other hand, it can alter the observed radio environment;
- high computational capabilities at the cognitive radios are required;
- the shared observations/decisions are “local” and could be corrupted by shadowing as discussed in Sect. 3.1 and can affect the performance of distributed spectrum sensing. In order to overcome such limitation, each terminal can associate a measure of accuracy (or confidence) to its shared information at the cost of an increase in dedicated channel bandwidth.

In order to overcome the above listed issues, distributed detection without fusion spectrum sensing techniques have been developed. In these approaches a set of N cooperative cognitive radios share the same radio environment. Each cognitive radio performs spectrum sensing based on its local observation. These local decisions are not fused to obtain a global decision and no sharing of information is required. Cognitive radios operation is coupled (dashed line in Fig. 3) to obtain a single decision based on a global goal [42]. An example of distributed detection without fusion is shown in Fig. 3.

In Sect. 5.3 a detailed description of a distributed detection without fusion spectrum sensing technique based on distributed detection theory [42] will be provided. As will be shown, such a technique is based on “implicit” cooperation among terminals and it does not require any dedicated channel.

Fig. 3 Example of distributed detection without fusion scenario: N cognitive radios share the same radio environment and observe it. Each cognitive radio performs local decision (u_i , $i = 1, \dots, N$). The cognitive radio do not communicate with each other, but their operation is coupled (dashed line) to obtain a global goal



4 Embodied Cognition-Based Systems: Their Role in Cognitive Radio

The algorithmic solutions for spectrum sensing (or often called mode identification and spectrum monitoring—MISM) developed until now, are only a part of the whole cognitive radio system. As a matter of fact, there are operative methodologies which describe the entire behavioral model, the so-called CC. These methodologies could be adapted for the cognitive radio system.

In order to introduce the proposed embodied cognition-based framework, let us recall the basic characteristics of a CC. The first stage of the cycle (Sensing or Observation) represents a passive interaction of the terminal with the environment: the cognitive terminal (CT) gathers information about both its internal state and the surrounding environment. In the second step (Analysis), the acquired data are processed and analyzed in order to provide the system with a representation of the perceived context. In the Decision stage, the cognitive system has to decide which is the most proper (re)action to the received coupled external/internal stimuli (i.e., a contextual response). The action represents an active interaction with the external environment because the CT tries to influence the physical context through its actions, in order to gain an “advantage.” In engineering terms, the CT evaluates a functional that represents a cost/merit related to a certain potential action.

While the CC is a shared concept among almost the entire cognitive radio community, different research lines can be seen in how the knowledge is managed and processed within each stage of the cycle. In fact, each stage of the CC requires management of information, which can be *naturally* embedded in the entity itself or acquired during its normal *life*. This knowledge can be organized according to two principal models:

- Symbolic Representation—Semantic Inference.
- Physically (body) grounded signal-based representation.

The former model tries to describe the knowledge in the classical rule-based approach for the construction of AIs [30]. The latter, and perhaps more interesting,

vision takes inspiration from the work on Robotics of Brooks [5] and looks at intelligence as emerging from the active (interactional) body capabilities, the basic one of which is surely the possibility of motion. It is referred in the literature as *Embodied Cognition* [38]. A confirmation, at a biological level, of the validity of this approach to intelligence comes from recent neurophysiological studies: the neuroscientist Llinas [27] hypothesized that, from an evolutionary point of view, one of the primary goal of intelligent multicellular organisms evolving toward higher level organisms is to use contextual information obtained through sensing to move in the surrounding environment. Motion can provide to the living entity an advantage in life conditions, due, for example, to the ability to reach a safer or a food-rich point. In the human brain, these kinds of motion, which are genetically codified into the human being, are generated by specific groups of neurons called *fixed action patterns* (FAPs), whose output is able to modulate motor muscles actions.

These preliminary assumptions can lead to the definition of cognitive models (and hence of specific CCs) characterized by specific *embodied* features that are particularly useful in the design process of cooperation mechanisms, such as distributed spectrum sensing.

The representation of the internal knowledge in *embodied* systems, and hence the description of context, is hence strictly linked with the perceptive/motory possibilities of the entity itself. Physical limitations of motion possibilities drive, tackling back the CC, the possibilities at the decision stage too and its internal knowledge representation. The same concept can hence be extended going backward into the cycle until to the sensing stage.

This fact is evident when the cognitive radio is considered as a subcomponent of a more general cognitive system, like a mini-robot which can move independently in a known or unknown environment. In this situation, the Decision stage can be tuned in order to move the robot (or to suggest a motion to a human) to a location which allows the best “point of view” for spectrum monitoring and analysis. If at least two cognitive cooperative entities are present in the environment, a distributed algorithm for transmission mode classification can be developed starting from the embodied formalization and management of knowledge.

However, a coherent problem definition together with a knowledge representation is needed to allow a quantitative engineering approach. In the following, these concepts, ranging from general knowledge representation to specific analysis and decision tools needed to provide a suitable framework, are presented.

5 Spectrum Sensing in Embodied Cognition-Based System

5.1 Spectrum Sensing Problem Definition

Let us consider a set of CTs $CT = \{CT_n : n = 1, \dots, N\}$ moving in an environment characterized by the presence of a set of radio sources $RS = \{RS_k : k = 1, \dots, K\}$. Each source is defined by the pair $RS_k = \{x_{RS_k}, Md_m\}$ where x_{RS_k} is the position of

the single source in the sources' common reference system \underline{X}_{RS} and $Md = \{Md_m : m = 1, \dots, M\}$ defines one of the possible air interfaces considered in the problem. The single air interface is hence defined by $Md_m = \{\text{Mod}^m, B^m, C^m, P_t^m\}$ where Mod^m is the transmission modality (e.g., Orthogonal Frequency Division Multiplexing (OFDM), Complementary Code Keying (CCK)), B^m is the occupied bandwidth, C^m is the central carrier frequency and finally P_t^m is the transmitted power level.

According to the above notation, the general problem addressed in this paper can be defined as finding the active air interfaces $Md_m, m = 1, \dots, M' \leq M$ associated with a set of active radio sources $RS_k, k = 1, \dots, K' \leq K$ starting from spectrum sensing performed by the pool of N cooperating CTs which embed embodied cognition capabilities.

5.2 Embodied Cognitive Sensor Definition

The Embodied knowledge representation is derived from the physical body capabilities. For this reason, it is necessary to define the body of the CT first. Ideally, the definition should be in terms of mass/volume/inertia, but here it will be related to the CT's interactive capabilities which influence the behavior of the CT itself.

Let us define the body through its environmental interactive aspects, i.e., it is equipped with a set of sensors $Se = \{Se_h : h = 1, \dots, H\}$ and it can perform a set of possible actions $A = \{A_p : p = 1, \dots, P\}$ in the space.

As a starting point, let us define a basic cooperative body where $Se_c = \{Se_h : h = R, V\}$ with R and V representing the radio (omnidirectional antenna) and video sensing modalities, respectively, and the index c refers to the cooperative terminal.

The possible actions, at time t , are $A = \{\Delta x = (\rho \cos(\theta), \rho \sin(\theta)) : 0 \leq \theta \leq 2\pi, \rho = \text{constant}\}$, i.e., they are limited to an omnidirectional movement, of the body itself, of constant length ρ . A basic FAP (Sect. 4) can hence be defined as $FAP = \{\Delta x(t) : t = t_0, \dots, t_0 + \Gamma T\}$, where Γ is the maximum number of successive actions to pursue, while T is the discrete time interval.

Besides the body, each CT can also be defined through the knowledge, embodied in itself, that allows it to function. Let us consider a homomorphic set of terminals where each terminal is supposed to have the same behavioral model (e.g., a set of "cloned" robots).

Starting from the basic body, the required knowledge can be defined as the set $K_n = \{K_{P_n}, K_E, K_{Env}\}$, where K_{P_n} is the knowledge about the space surrounding the terminal, K_E is composed of all the *embodied* functions that constitute the CC, and K_{Env} is the knowledge that the CT has available about the physical/statistical interaction characteristics of the objects present in the environment.

The space surrounding each cooperating terminal is referred to via its own reference coordinate system (RCS) and hence $K_{P_n} = \{x_{CT_n}(t) \in \underline{X}_{CT_n} : t = t_0, \dots, t_0 + \omega T\}$, where ω is the running time index.

The Embodied knowledge K_E can be structured into two levels for each component r of the CC: $K_E = \{E^r(\underline{X}_{CT_n}), F^r(\cdot) : r = \{\text{Sense, Analyze, Decide, Act}\}\}$. The

Table 2 Instinctual knowledge codification

$E^r(\underline{X}_{CT_n})$	Description
$E^{\text{Sense}}(\underline{X}_{CT_n})$	Antenna radiation pattern, video-camera field-of-view
$E^{\text{Analyze}}(\underline{X}_{CT_n})$	Available algorithms/methodologies (time/space/condition choice)
$E^{\text{Decide}}(\underline{X}_{CT_n})$	FAPs library—all the possible motion strategies
$E^{\text{Act}}(\underline{X}_{CT_n})$	Physical driver signals (voltage, current, time)

first level $E^r(\underline{X}_{CT_n})$ is composed of all the *naturally* embedded knowledge codified into embodied maps. The attribute *embodied* related to the representation of the knowledge means that it is all referenced to the CT's body or the CT's point of view. In Table 2 a more detailed explanation the maps for each stage of the CC is shown.

The second level is the procedural knowledge represented by the operational basic functions that constitute the interstage information transformation processes within the CC. Given a certain environmental context $E_c(t)$, how this condition of the external world is processed within the CT in Table 3 is shown.

Let us remark that in the present work only the design process of the Analysis and Decision functions will be considered. Furthermore, the semantic or subsemantic label $L(t, x_{CT_n})$ is one out of all the possible C' labels considered for the spectrum sensing problem, i.e. detecting the active air interfaces Md_m of the active radio sources RS_k .

The environmental knowledge can be generally defined as $K_{Env} = \{K_{PE}^i, K_{BE}^i : i = 1, \dots, I\}$, i.e., it contains the information about the relative position (K_{PE}^i), with respect to the CT, and the behavioral model (K_{BE}^i) of all the I entities interacting with the CT itself. Let us suppose that the CT always perceives the other interacting entities as cognitive (see Fig. 4). This concept will be used in the following as the basis for designing the simulation tool, since each one of the interacting entities is described through the sets contained in K_{Env} while the environment is substantially

Table 3 Procedural embodied knowledge

Functions	Description
$O(t, x_{CT_n}) = F^{\text{Sense}}(x_{CT_n}, E_c(t), K_n)$	Extraction of the observation set $O(t, x_{CT_n}) = \{O_n^h(t, x_{CT_n}) : h = 1, \dots, H\}$
$v(t, x_{CT_n}) = v(x_{CT_n}, K_n, O(t, x_{CT_n}))$	Features extraction (encapsulated in the Analysis Stage)
$L(t, x_{CT_n}) = F^{\text{Analyze}}(x_{CT_n}, O(t, x_{CT_n}), K_n)$ $= C(x_{CT_n}, K_n, v(x_{CT_n}, K_n, O(t, x_{CT_n})))$	Label extraction – Context Representation Classification + Feature Extraction
$D(t, x_{CT_n}) = F^{\text{Decide}}(x_{CT_n}, L(t, x_{CT_n}), K_n)$	Taken Decision
$a(t, x_{CT_n}) = F^{\text{Act}}(x_{CT_n}, K_n, D(t, x_{CT_n}), O(t, x_{CT_n}))$	Choice of the motion direction Pilot signals generation

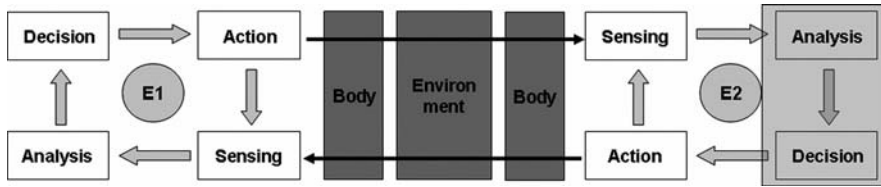


Fig. 4 Cognitive entity E1 can represent the cognitive entity E2 through a set of active *mirror* knowledge simulating a cognitive cycle [28]

translated into a virtual shared medium, where all the players *read/write* performing their sensing/actions.

The specific environmental knowledge can finally be represented as

$$K_{Env} = \{K_{P_{Room}}, K_{B_{Room}}, K_{P_{RS}}^k, K_{B_{RS}}^k, K_{P_j}, K_{B_j} : k = 1, \dots, K, j = 1, \dots, N, j \neq n\} \quad (1)$$

where each component is analyzed in Table 4.

In the following, a specific formalization for a spectrum sensing embodied algorithm will be presented.

5.3 Distributed Embodied Cognition Approach

In this section, a general architecture for the Analysis and Decision stages of an embodied cognitive radio terminal is presented. We leave out the interactional parts (i.e., the sensing and action stages), under the assumption of a physically ideal body.

Table 4 Components of the environmental knowledge

Knowledge	Definition	Description
$K_{P_{Room}}$	$\{\underline{X}_{Room}, T_{Room}^n(\underline{X}_{CT_n}, t) : \underline{x}_{CT_n}^{Room}(t) = T_{Room}^n(\underline{x}_{CT_n}(t), t)\}$	Information (reference coordinate system—RCS, Transformation Function) required for computing the absolute position of the CT in the environment (or <i>Room</i>).
$K_{B_{Room}}$	–	Behavior of the <i>Room</i> (e.g. walls, doors)
$K_{P_{RS}}^k$	$K_{P_{RS}}^k = \{\underline{x}_{RS_k}^n(t) \in \underline{X}_{CT_n}, \underline{X}_{RS_k}, T_{RS_k}(\underline{X}_{RS_k}, t)\}$	Relative position/orientation of the j -th radio source; transformation function for linking with <i>Room</i> RCS.
$K_{B_{RS}}^k$	–	Statistical description of the feature distributions for all the possible transmission situations all over the <i>Room</i> .
K_{P_j}	$\{\underline{x}_{CT_j}^n(t) \in \underline{X}_{CT_n}, \underline{X}_{CT_j}, T_{CT_j}(\underline{X}_{CT_j}, t)\}$	Relative position/orientation of the j th CT respect to the n th one (whose point of view is under analysis) and transformation functions respect to n -th RCS.
K_{B_j}	–	Under homomorphicity assumption is a <i>mirroring</i> (as previously seen for interactions) of the Embodied knowledge of n .

Let us start by assuming that the *Room* and the radio sources can be managed as only one single, more complex, entity. In fact, the features the CT can observe are the results of the interaction between the electromagnetic field (e.m.) field emitted by the radio sources and the physical component of the *Room* (e.g., the multipath effect).

As a starting point, let us consider a simple framework where only two CTs and only one radio source (able to communicate with the Md_1 air interface) that can be switched on (hypothesis H_1) or switched off (hypothesis H_0) are present in the environment.

After re-writing $x_{CT_i}^{Room}(t) = x_i(t)$, let us now define the quantities involved in the information processing within the CC for CT_1 (simply extensible to CT_2), starting from y_i , that are the features extracted by the i th CT from the radio signal perceived by the RS. The probability density functions (pdfs) $p(y_i|H_0, x_i)$ and $p(y_i|H_1, x_i)$ statistically describe how the RS influence the perceptions of the CT_i in both the possible cases. Generalizing the previously defined pdfs, it is possible to obtain a general behavior of the perceptual interactions between the CT and the RS in all the *Room*: K_{BRS}^1 comprises $p(y|H_0, x)$ and $p(y|H_1, x)$. The vector of features, that each CT extracts from its observations, is hence composed of $v(t, x_1) = \{y_i, \hat{x}_1, \hat{x}_2, \Delta x_2\}$, where the *hat* represented an estimation of the considered variable.

Compared to the framework presented in [7, 17], in the current work the hypothesis of perfect location knowledge is relaxed. The two variables are considered as estimated by proper feature extraction functions, each one related to a pdf that describes the statistical behavior of the function itself. The random variable (rv) \hat{x}_1 is hence described by the pdf $p(x_{CT}|\hat{x}_{CT_1})$, while the rv \hat{x}_2 requires further analysis, \hat{x}_2 being a function of \hat{x}_1 . Let us consider, as an example, a transformation function \hat{T}_{CT_1} composed of the only translational component. Calling \hat{d}_2 the estimated distance vector between CT_1 and CT_2 , the absolute positions of the two CTs are linked through the relationship $\hat{x}_2 = \hat{x}_1 + \hat{d}_2$. In this simple case, $p(x_1|\hat{x}_1)$ being the pdf of \hat{x}_1 and $p(d_2|\hat{d}_2)$ the pdf of \hat{d}_2 , the pdf of \hat{x}_2 will be $p(x_2|\hat{x}_2) = p(x_1|\hat{x}_1) * p(d_2|\hat{d}_2)$, where $*$ denotes the convolutional operator.

Let $u_i = j : j = \{0, 1\}$ be the classification performed by CT_i about the presence of the hypothesis H_0 or H_1 . This classification is associated with the presence of the air interface Md_1 . It is hence possible to infer that the u_i represents the MISM classification. The pdfs $p(u_i = j|y_i) : j = \{0, 1\}$ describe the statistical behavior of the MISM classification algorithm in relationship with the perceived features y_i . This knowledge is part of the behavioral model of the interacting entity K_{B_2} , but, under the homomorphic assumption, it is also a part of the embodied knowledge, i.e. it is in $E^{Analyze}$.

Each CT estimates the behavior of the companion CT through a mirror (or inverse) decision process. Let us call this estimate \hat{u}_i . The context label can hence be defined as $L(t, x_1) = \{u_1, \hat{u}_2, x_1, x_2\}$.

Once the most important variables are defined, it is possible to analyze in more detail how the single stages of the cycle can be structured.

5.3.1 Analysis Stage

The analysis stage is shown in Fig. 5. The feature extractor corresponds to the function $v(x_{CT_n}, K_n, O(t, x_{CT_n}))$ defined in Sect. 5.2 and extracts from the perceived observations the vector of features described in Sect. 5.3.

In particular, distributed detection is the fundamental component of the analysis stage of the CC for the embodied cooperative CTs. This fact will be more clear after the introduction of the distributed detection theory applied to the considered MISM problem.

Starting from basic Distributed Detection Theory [42] and its application to an ideal (in terms of location knowledge) MISM problem [7, 17], it is possible to re-formalize the theory, in order to keep into account the uncertainty introduced by the estimated locations.

Let us now define the distributed classification function [42]

$$\Lambda(y_1, x_1) = \sum_{u_1=0}^{u_1=1} t_1(x_2) \tag{2}$$

where Λ is the classical Bayesian likelihood function [24], u_n represents the classification performed by the n th CT, and t_n is the distributed detection threshold [42].

The likelihood function has to be arranged in order to manage the uncertainty of \hat{x}_1 :

$$\Lambda(y_n, \hat{x}_n) = \frac{\int_{X_{Room}} p(y_n|H_1, x_n)p(x_n|\hat{x}_n)}{\int_{X_{Room}} p(y_n|H_0, x_n)p(x_n|\hat{x}_n)} \tag{3}$$

while the distributed classification threshold is now computed as

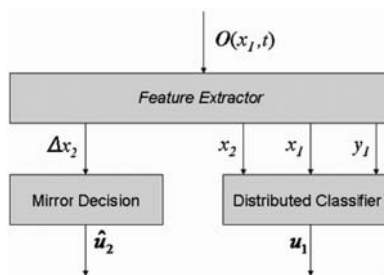


Fig. 5 Analysis module of an embodied CT. The analysis stage is composed of three main sub-blocks: the feature extractor, the mirror decision block, and the distributed classifier

$$t_n(\hat{x}_j) = \frac{P_0}{P_1} \cdot \frac{(K_d - 1) + (2 - K_d) \int_{\underline{X}_{Room}} p(u_j = 0|H_0, x_j)p(x_j|\hat{x}_j)}{1 + (K_d - 2) \int_{\underline{X}_{Room}} p(u_j = 0|H_1, x_j)p(x_j|\hat{x}_j)} \quad (4)$$

If the pdfs $p(x_1|\hat{x}_1)$ and $p(d_2|\hat{d}_2)$ are unknown, a possible approach for practical implementation can be their substitution with weighting functions $w(\hat{d}_2) : \Psi \rightarrow \mathbb{R}$ and $w'(\hat{x}_1) : \Pi \rightarrow \mathbb{R}$ where the domains Π and Ψ are limited portions of \underline{X}_{Room} .

It should be noted that the Bayesian threshold computed, based on distributed detection theory, incorporates both the statistical behavior of the RS and the classification behavior of CT_2 computed at the point \hat{x}_2 . This fact corresponds to the internal simulation of the CC of the interacting entities that each embodied CT should perform. In fact, the pdf $p(y_2|H_j, x_2)$ describes the perceptual interaction of CT_2 with the radio sources, while the pdf $p(u_2 = j|y_2, x_2)$ represents the Analysis stage of the companion CT. This is one of the main reasons why the distributed detection theory perfectly fits within the embodied cognition framework described here. This theory paradigm allows us to simulate the behavior of the interacting entities and to compare it with the observations/classification each CT performs (represented by the likelihood function) in a one-shot computation, with a low computational load compared to other solutions (e.g., agent-based internal emulation).

This simple binary case can be extended to multiple radio sources/air interfaces as shown in [7, 17] by using binary tree-based parameters selection for a single binary classifier or a multiple classifier approach where each classifier tests the presence or absence of a specific hypothesis. The Mirror Decision stage is used in the analysis stage to estimate which class CT_2 could have classified having the action (motion) the CT has actuated as input. This block will not be here considered, but let us postpone some general considerations about it until after the introduction of the Decision stage of the Embodied CTs.

The Distributed Classifier and the Mirror Decision, that work in parallel, comprise the classification function $C(x_{CT_n}, K_n, v(x_{CT_n}, K_n, O(t, x_{CT_n})))$, defined in Sect. 5.2, and together with the feature extractor above described they form the Analysis survival function F^{Sense} .

5.3.2 Decision Stage

Before describing the Decision stage, it is necessary to define the final goal for the CT. From a physiological point of view, the final goal of a living entity is *homeostasis* [6], i.e., reaching of a dynamic equilibrium that allows the life of the entity itself. This concept can be extended to higher cognitive layers: it is possible to infer that *homeostasis* is the status of the CT in which it has gained the maximum advantage (as previously cited in Sect. 4) with respect to its physical possibilities and to the environmental context. In engineering terms, this situation corresponds to having reached a maximum/minimum of a merit/cost functional. In the case of the MISM

system presented here, the functional should evaluate how much value the position of the CT provides in the e.m. context.

The decision stage is devoted to choosing the best way to reach an homeostatic situation. An engineering translation for this concept can be the minimization of a global cost functional

$$x_T = \arg_{x_i} \min_{x_1, x_2} J(x_1, x_2, u_1, u_2) \quad (5)$$

where x_T is called the *target point* and represents the point where the CT can reach its dynamic equilibrium. Since u_2 is unavailable to CT_1 it is possible to use a suboptimal version

$$x_T = \arg_{x_i} \min J(x_1, x_2, u_1, \hat{u}_2) \quad (6)$$

by replacing the u_2 with its estimated version.

The decision stage choses the FAP that leads to the current dynamic target point x_T in a more direct (or with minimal effort) way. This is possible through the recursive usage of a deterministic look-up table $LT(u_i) \rightarrow \Delta x$ that associates a motion $\Delta x(x_i)$, parametrized by the position of the CT, for each classification performed by the CT. This table should be invertible, hence it should be possible to define $LT^{-1}(\Delta x(x_i)) \rightarrow \hat{u}_i$. This inverse function can be used in the Mirror Decision block of the analysis stage as estimator for the class decided by the companion CT.

In the following, how the proposed architecture can perform in a particular simulated case will be presented.

6 Simulation and Results

6.1 Simulation Framework

Let us define the characteristics of the MISM problem, starting from the basic one, addressed here. The simulated framework is presented in Fig. 6. The chosen Mds have the particular characteristic that they share the same bandwidth and they can operate simultaneously with their signals superimposed on each other. Furthermore, Bluetooth (BT) transmits with a very low power (1 mW) within a range which is much more limited than the WiFi (or WLAN—wireless local area network) one.

Each source can be associated with only one Md and the transmitted signal is affected by the typical propagative phenomena that can be found in a common office, according to the model presented in [35]. The possible situations the CT could find in the environment are represented by four classes: WLAN, when only the WLAN RS is switched on; BLUE, when only the BT RS is switched on; WLBL, when both the RSs are switched on; NOISE, when only environmental noise is present. The two CTs involved in the MISM classification can enter the room at any point of the perimeter, without the consideration of the presence of fixed doors/walls, and they

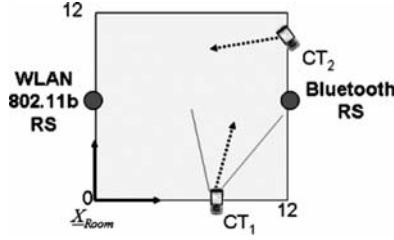


Fig. 6 Simulated MISM Problem. Two RSs are present in a Room of 12×12 m. The absolute RCS X_{Room} has its origin in one of the corners of the room. Two communication modes are possible: $Md_1 \rightarrow$ IEEE 802.11b WiFi; $Md_2 \rightarrow$ Bluetooth

are able to move within the room itself. Each CT is considered provided with the basic cooperative body defined in Sect. 5.2 with $\rho = 1m$ and $\Gamma = 1$. Each FAP is composed of a single motion Δx .

In order to solve the specific MISM problem, two time–frequency (TF) features, derived from the Wigner–Ville TF transform [11], have been used. In particular, the standard deviation of the instantaneous frequency (σ_ω) and maximum time duration of the signal (T_{max}) have been proved [17] to be useful in the case of signals superimposed in the same bandwidth.

The vector y_i is hence composed of $y_i = [\sigma_\omega T_{max}]$ and it is used as input for the distributed classifier, together with the instantaneous positions of the two CTs. In order to address the multiclass MISM problem, the multiple classifier One-Against-All architecture [7] has been chosen. In the proposed implementation, this architecture requires the computation of the upper bound of the theoretical error probability. This information is obtained through simulated sample means and covariances of the classes. Under the assumption of Gaussian $p(y_i|H_j, x_i) : j = \{WLAN, BLUE, WLBL, NOISE\}$, it is possible to compute the Bhattacharyya distance [4] and the Chernoff bound [9] $C_{j,k}(x_i)$ for each pair of classes. The upper bound for a selected class j , $P_{err}^j(x_i)$, is hence given by $P_{err}^j(x_i) = \max_{k, k \neq j} C_{j,k}(x_i)$. The values obtained at different training points of the environment have been interpolated in order to obtain a continuous surface all over the room $P_{err}^j(x)$. In order to design the decision stage, the global minimum of $P_{err}^j(x)$, for each class j , has been chosen as the target point where the CT can reach the homeostatic condition $x_T^j = \arg \min P_{err}^j(x)$. It is hence easy to obtain the look-up table $LT(u_i)$ (see Table 5). The chosen motion is hence parametrized by x_i before passing to the action stage that will translate it into control signals (not considered in the present paper):

$$\begin{aligned} \theta &= \angle(x_T^j - x_i) \\ \Delta x(x_i) &= [\cos(\theta) \sin(\theta)] \end{aligned} \quad (7)$$

The inverse decision stage is hence defined as

$$\hat{u}_i(t^* - T) = \arg \max_j \Delta \mathbf{x}_i(t^*) \cdot \mathbf{x}_T^j(x_i, t^* - T) \quad (8)$$

Table 5 Lookup table of the decision stage

u_i	Δx
WLAN	move to x_T^{WLAN}
BLUE	move to x_T^{BLUE}
WLBL	move to x_T^{WLBL}
NOISE	move to x_T^{NOISE}

where $\Delta \mathbf{x}_i(t^*)$ is the unit vector of the motion vector of CT_i , perceived by the other CT at the time instant t^* while $\mathbf{x}_T^j(x_i, t^* - T)$ is the unit vector of the direction that links the previous position of CT_i with the target point of the j th class.

As weighting functions w and w' , two equal 2D rectangular functions have been used. The width of each function is $1 \times 1 \text{ m}^2$.

The developed simulation architecture is presented in Fig. 7 . Each CT has been developed as a dynamic system through a closed-loop finite state machine whose data structures are organized in the same sets as described in Sect. 5.2. Apart from the specific implementation language, the CT has been implemented in order to obtain an “emulation” of the cognitive core of the system. In fact, with the proper language-dependent adjustments, it is possible to export the same architectural structure on a hardware platform, without any particular ad hoc modifications. Furthermore, the embodied framework is so general that it is possible to add new functionalities (or to improve/modify the existing ones) without any impact on the simulation structure itself.

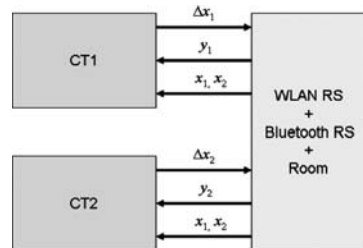
In the following, the results obtained with the simulative/emulative system will be presented.

6.2 Results

In order to simulate the uncertainty introduced by the sensing-based localization, a 2D Gaussian noise has been added to the absolute positions of the CTs, according to the definition of the simulated problem. Furthermore, the following simulation parameters have been used:

- Maximum number of iterations per simulation: 1000
- Number of simulations per class per problem: 1000

Fig. 7 Simulation architecture. The simulation system has been developed in Matlab/Simulink[®] and it has been built up according to the organization of knowledge and the interaction models presented in Sect. 5.2



- Standard deviation of positioning uncertainty: $\rho_x, \rho_d = \{0m, 1m, 2m\}$
- $K_d = \{2, 5\}$
- uniformly random choice for the entrance points of the CTs in the room

In the conditions considered, the simulator is able to perform a complete CC in about 0.019 s on a 1.86 GHz Intel Core 2[®] equipped general purpose PC with 1 GByte DDR2 RAM.

Results for $K_d = 2$, that represent substantially a stand-alone classification, will be omitted because they further confirm what was shown in [7] and [17]. As a matter of fact, results obtained prove the effectiveness of the distributed method compared with the stand-alone one.

Let us now consider the distributed algorithm with $K_d = 5$ (focus of the present chapter) and let us analyze its behavior with the generalized computation of the likelihood function and of the threshold. Results will be displayed in the form of confusion matrices, where the first column indicates the *ground truth* (GT), i.e., the real contextual situation, while the other columns represent the distribution of the classifications performed by the CT. In the following, for simplicity of reading, the class labels will be further abbreviated as W (WLAN), B (BLUE), WB (WLBL), and N (NOISE).

An error on the localization of the companion terminal has been introduced. As previously mentioned two cases has been evaluated: Gaussian errors with 1m or 2m of standard deviation. The introduction of these uncertainties, partially compensated by the weighting function, has little impact on the performances of the classifier as shown in Table 6 and in Table 7. A better design of w could lead to a substantial reduction of these errors. The introduction of the weighting function w' , in the computation of both Λ and t_i , has an unexpected results on the robustness of the system. In fact, despite the error introduced in the self localization of the terminal, the obtained results are similar to the ones obtained with perfect knowledge.

Since we are interested to evaluate the system performances in the worst conditions, only variances of 2m will be considered.

Probably more interesting results are provided by the simultaneous usage of the distributed classifier and of the embodied cognition framework. The previous results are referred to one-shot classifications, while the embodied cognition framework employs an iterative approach which has as its goal reaching the homeostatic condition. For this reason, let us analyze the performances of mode classification in the homeostatic condition. The confusion matrices of the final mode classification at

Table 6 Confusion matrices for $\rho_x = 0m$

$\rho_x = 0m$	$\rho_d = 1m$				$\rho_d = 2m$			
	W (%)	B (%)	WB (%)	N (%)	W (%)	B (%)	WB (%)	N (%)
GT/CLASS								
WLAN	79.1	7.4	13.3	0.2	79.9	7.6	12.3	0.2
BLUE	30.5	68.5	0.0	1.0	30.6	68.6	0.0	0.8
WLBL	79.1	20.1	0.7	0.1	79.1	20.1	0.7	0.1
NOISE	0.0	41.2	0.0	58.8	0.0	40.6	0.0	59.4

Table 7 Confusion matrix for $\rho_x = 2m$

$\rho_x = 2m$	$\rho_x = 2m$			
	W (%)	B (%)	WB (%)	N (%)
GT/CLASS				
WLAN	85.8	11.2	2.8	0.2
BLUE	0.7	76.7	1.2	21.4
WLBL	63.2	30.4	6.3	0.1
NOISE	0.0	10.7	0.0	89.3

Table 8 Confusion matrices in homeostatic situation

$\rho_x = 2m$	$\rho_d = 0, \rho_x = 2$ —homeostasis				$\rho_d = 2, \rho_x = 2$ —homeostasis			
	W (%)	B (%)	WB (%)	N (%)	W (%)	B (%)	WB (%)	N (%)
GT/CLASS								
WLAN	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
BLUE	0.0	100.0	0.0	0.0	0.0	100.0	0.0	0.0
WLBL	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
NOISE	0.0	0.7	0.0	99.3	0.0	0.2	0.0	99.8

homeostasis, for the most complex simulated problems, in Table 8 are shown. We observe that, even with memoryless terminals, the iterative exploration of the room and the reciprocal observation of the two involved CTs, lead to an almost complete reduction of the MISM misclassification, with the only exception of the WLBL class. As a matter of fact, in an office indoor environment, BT has a maximum range of less than 10m, and the combined effects of multipath and the spurious cross-terms introduced by the Wigner distribution can create misclassifications. This exception is relatively problematic for a MISM application. In fact, the CTs always decide the presence of an available communication signal and never confuse it with the NOISE class. These errors hence do not limit the *always on* communication capabilities of the CT.

7 Conclusions

This chapter introduced the methodology to solve the autonomic computing problems with CR terminals. A discussion on the state of the art on the different visions on how to build up a CR and how different spectrum sensing capabilities can be implemented, has been provided. Then an embodied cognition-based approach to distributed spectrum sensing was presented. The new framework was designed by combining knowledge coming from different fields of cognitive neurosciences, robotics, and AI. Starting from the awareness of the physical capabilities of the body, the mind of the embodied cognitive radio system can be developed through an organization of the internal knowledge that directly represent the active/passive interactions of the entity with the players involved in the problem. This framework has been particularly designed addressing the MISM problem and some mathematical tools, fitting with the framework, for the implementation of such a system have

been provided. In particular, Distributed Detection Theory has been shown to realize embodied cognition capabilities in a simple and fruitful way.

A simulation/emulation tool has been implemented in order to prove the effectiveness of the embodied cognition framework. The simulated problems were intended as the first steps in the direction of the applicability of distributed spectrum sensing to more realistic and more complex autonomic computing problems compared to the ones that could be found in the state of the art.

References

1. Akyildiz I, Lee W, Vuran M, Mohanty S (2006) NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer Networks* 50(13):2127–2159
2. Azzouz E, Nandi A (1996) *Automatic Modulation Recognition of Communication Signals*. Kluwer Academic Publishers Norwell, MA, USA
3. Bai L, Chou D, Yen D, Lin B (2005) Mobile commerce: its market analyses. *International Journal of Mobile Communications* 3(1):66–81
4. Bhattacharyya A (1943) On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society* 35:99–109
5. Brooks RA (1991) Elephants do not play chess, MIT press, chap in “Designing Autonomous Agents”, pp 3–15
6. Canon WB (1932) *The wisdom of the body*. W. W. Norton & Co., New York, USA
7. Cattoni A, Minetti I, Gandetto M, Niu R, Varshney P, Regazzoni C (2006) A spectrum sensing algorithm based on distributed cognitive models. In: *SDR Forum Technical Conference*, Orlando, FL, USA
8. Chen CK, Gardner WA (1992) Signal-selective time-difference-of-arrival estimation for passive location of man-made signal sources in highly corruptive environments, Part II: algorithms and performance. *IEEE Transactions on Signal Processing* 40(5):1185–1197
9. Chernoff H (1952) A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics* 23(4):493–507
10. Clancy C, Hecker J, Stuntebeck E, O’Shea T (2007) Applications of machine learning to cognitive radio networks. *IEEE Wireless Communications* 14(4):47–52
11. Cohen L (1994) *Time Frequency Analysis : Theory and Applications*, 1st edn. Prentice-Hall Signal Processing, Prentice Hall PTR
12. da Silva CRCM, Choi B, Kim K (2007) Distributed Spectrum Sensing for Cognitive Radio Systems. In: *2007 Workshop on Information Theory and Applications*, La Jolla, CA, USA
13. DARPA XG WG (2003) *The XG Architectural Framework v1.0*. Tech. rep., DARPA
14. Dobre O, Abdi A, Bar-Ness Y, Su W (2007) Survey of automatic modulation classification techniques: classical approaches and new trends. *IET Communications* 1:137–156
15. FCC (2002) *Spectrum policy task force report*. Tech. rep., Federal Communication Commission
16. Fette B (2006) *Cognitive Radio Technology*. Elsevier-Newnes, Burlington, MA, USA
17. Gandetto M, Regazzoni C (2007) Spectrum sensing: a distributed approach for cognitive terminals. *IEEE Journal on Selected Areas in Communications* 25(3):546–557
18. Gandetto M, Guainazzo M, Regazzoni CS (2004) Use of time-frequency analysis and neural networks for mode identification in a wireless software-defined radio approach. *Eurasip Journal of Applied Signal Processing* 13:1778–1790
19. Gandetto M, Cattoni A, Regazzoni CS (2006) A distributed wireless sensor network for radio scene analysis. Taylor and Francis Publishing – *International Journal of Distributed Sensor Networks* 2(3)

20. Gardner W (1980) A unifying view of second-order measures of quality for signal classification. *IEEE Transactions on Communications* 28(6):807–816
21. Gardner WA (1988) Signal interception: a unifying theoretical framework for feature detection. *IEEE Transactions on Communications* 36(8):897–906
22. Godard L, Moy C, Palicot J (2006) From a configuration management to a cognitive radio management of SDR systems. In: *Cognitive Radio Oriented Wireless Networks and Communications*, pp 1–5
23. Haykin S (2005) Cognitive radio: brain-empowered wireless communications. *IEEE Journal on Selected Areas in Communications* 23(2):201–220
24. Fukunaga K (1990) *Introduction to Statistical Pattern Recognition*, second edition edn. Academic Press Inc., New York
25. Kephart JO, Chess DM (2003) The vision of autonomic computing. *Computer* 36(1):41–50
26. Kolodzy P (2001) Next Generation communications: Kickoff meeting. In: *Proc. DARPA*
27. Llinas R (2001) *I of the Vortex*. Bradford Book, MIT Press, Cambridge, MA
28. Marchesotti L, Piva S, Regazzoni CS (2005) Structured context-analysis techniques in biologically inspired ambient-intelligence systems. *IEEE Trans on Systems, Man, and Cybernetics - Part A : Systems and Humans* 35(1):106–120
29. Mitola J (1999) Cognitive radio: making software radio more personal. *IEEE Pers Comm* 6(4):48–52
30. Newell A (1982) The knowledge level. *Artificial Intelligence* 18(1):87–127
31. Nolan K, Sutton P, Doyle L (2006) An encapsulation for reasoning, learning, knowledge representation, and reconfiguration cognitive radio elements. In: *Cognitive Radio Oriented Wireless Networks and Communications*
32. Oner M, Jondral F (2007) On the extraction of the channel allocation information in spectrum pooling systems. *IEEE Journal on Selected Areas in Communications* 25(3):558–565
33. Proakis JG (2001) *Digital Communications*, 4th edn. McGraw Hill, New York
34. Sahai A, Hoven N, Tandra R (2004) Some fundamental limits on cognitive radio. In: *Allerton Conference on Communications, Control and Computing*
35. Saleh A, Valenzuela R (1987) A statistical model for indoor radio propagation. *IEEE J Select Areas Commun SAC-5*:128–141
36. Schulze H, Luders C (2005) *Theory and Applications of OFDM and CDMA Wideband Wireless Communications*. John Wiley and Sons, Ltd. New York
37. Srinivasa S, Jafar SA (2007) The throughput potential of cognitive radio: a theoretical perspective. *IEEE Communications Magazine* 45(5):73–79
38. Steels L, Brooks R (1995) *The Artificial Life Route to Artificial Intelligence: Building Embodied Situated Agents*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ
39. Sutton P, Doyle L, Nolan K (2006) A Reconfigurable Platform for Cognitive Networks. In: *Cognitive Radio Oriented Wireless Networks and Communications*
40. Sutton P, Nolan K, Doyle L (2008) Cyclostationary signatures in practical cognitive radio applications. *IEEE Journal on Selected Areas in Communications* 26(1):13–24
41. Urkowitz H (1967) Energy Detection of unknown deterministic signals. *Proceedings of IEEE* 55(4):523–531
42. Varshney P (1996) *Distributed Detection and Data Fusion*, 1st edn. Springer-Verlag, Heidelberg

Autonomic Peer-to-Peer Systems: Incentive and Security Issues

Yu-Kwong Kwok

Abstract With voluntary users participating in an autonomic manner, peer-to-peer (P2P) systems have been proliferating in an unprecedented pace. Indeed, it is widely known that P2P traffic now constitutes over 60% of total Internet traffic. P2P systems are now used for file sharing, media streaming, and various other social networking applications. Furthermore, P2P systems are also extending their reach to the wireless realm. However, there are still two major system aspects that pose challenges to P2P systems' designers and users: incentives and security. First and foremost, a P2P system, by its nature, is viable only if users contribute their resources to the community. Obviously, uniform and global altruistic behaviors cannot be expected for all users.

Some users will definitely try to take advantage of the altruism of other participants. If such "free-riding" phenomenon is too wide spread, then a system collapse will result because the input to the P2P community is smaller than the output. Thus, it is important to incorporate effective incentive mechanisms to deter selfish behaviors and encourage active contributions. On the other hand, security related issues such as privacy, anonymity, and authentication are also beyond doubt critical concerns of the participating users. Essentially, users do not want to sacrifice security to trade for service. Clearly security issues and incentives are closely related in that a low security or "untrustworthy" P2P community will not attract a large population of contributing users, and instead, might even tempt malicious users to consider the system as a potential point of attacks.

In this chapter, we survey and analyze the current state-of-the-art in tackling the incentive and security issues in P2P systems. We first give a brief account of P2P applications and their wired and wireless operating environments. We then survey and critique existing incentive techniques. This is followed by the analysis of contemporary P2P security algorithms.

Y.-K. Kwok (✉)

Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80526-1373, USA

e-mail: Ricky.Kwok@colostate.edu

The techniques that we survey will range from traditional optimization algorithms to game theoretic schemes. We provide some of our thoughts on open research issues, followed by a conclusion.

1 Introduction

Advancements in computing and communication, coupled together, enable a recent trend in a new form of autonomic distributed processing—peer-to-peer (P2P) computing [59]. As its name implies, P2P computing involves users (or their machines) on equal footing—there is no designated server or client, at least in a persistent sense. Every participating user can be a server and be a client depending on context. Some people have referred this to as a “democratic computing environment” [23] because users are free from centralized authorities’ control. This new paradigm of distributed computing has spurred many high-profile applications, most notably in file sharing, with household names such as BitTorrent [30], Freenet [5, 29], Gnutella [7], and Napster [14].

According to a recent survey [19], P2P applications generate at least one-fifth of the total Internet traffic on a daily basis. It is believed that this trend will continue. Furthermore, an ISP (Internet Services Provider) solution company [17] reported that the hottest P2P applications are file-sharing applications, such as BitTorrent [2] (occupies 53% of all P2P traffic) and eDonkey2000 [3] (occupies 24%). Apart from these *wired P2P file sharing*, some other *wireless* P2P applications have become part of our daily life. For example, people can now play numerous P2P JAVA online games which are compatible with mobile phones so that players are allowed to interconnect in local area through Bluetooth or wide area through GSM/GPRS network (e.g., Nokia 6230, 7280, 6680, 3230, Sony Ericsson Z800i, Motorola E1000) [9]. Indeed, in many metropolitan cities such as Hong Kong and Tokyo, we can see that train commuters routinely play wireless games among each other using popular devices such as PSPs (Play Station Portables). Now, many mobile phones also already have 128 MB or higher RS-MMC (Reduced-Size MultiMedia Card) card storage capability. Indeed, it is now a common practice to have P2P file transfer through “BlackBerry e-mail service” on mobile phones. As such, wireless P2P file sharing is not only feasible but also becoming pervasive. We provide a brief survey of popular P2P applications in Sect. 2.

The highly flexible features of P2P computing such as a dynamic population (users come and go asynchronously at will), dynamic topologies (it is impractical, if not impossible, to enforce a fixed communication structure), and anonymity, come at a significant cost—autonomy, by its very nature, is not always in harmony with tight cooperation. Consequently, inefficient or lack of cooperation could lead to undesirable effects in P2P computing. Among them the most critical one is “free-riding” [36] behavior. Loosely speaking, free riding occurs when some users do not follow the presumed altruistic cooperation rules such as sharing files voluntarily, sharing bandwidth voluntarily, or sharing energy voluntarily, so as to benefit the

whole community. Incentives are needed to entice cooperation among peers to make the system work. We survey incentive techniques in Sect. 3.

Apart from incentives, system security is another important issue induced by the open and autonomic nature of a P2P network. Indeed, it is very difficult, if not impossible, to completely avoid having malicious peers participating in the network. Consequently, unlike many traditional systems, a P2P system has to tolerate many different attacks while still provides useful services. The bottomline is that the P2P system can still operate with a sufficiently large population of peers that are interested in participating, while taking up the risk of being attacked. We survey many security techniques in Sect. 4. We then also offer our suggested future research directions for the incentive and security issues in a P2P system in Sect. 5.

2 P2P Applications

In this section, we first present an overview of P2P applications, followed by a brief introduction of the most popular P2P application nowadays—BitTorrent [2]. We then describe the different architectures used in different contemporary P2P applications. We also briefly discuss the recent trend of running P2P applications on a wireless network.

Generally speaking, there are two mainstream applications in P2P environments: sharing of discrete data, and sharing of continuous data. Examples of the former include file-sharing systems (e.g., Napster [14]), data-sharing systems (e.g., sharing of financial or weather reports), etc. A notable example of the latter is P2P video streaming (e.g., PPLive [15]).

There is an important difference between file sharing and media streaming systems. In the former, a user needs to wait until a file (or a discrete unit of shared information) is completely received before it can be consumed or used. Thus, there could be a significant delay between service request and judgement of service quality. In an extreme case, a user may not discover that a shared file is indeed the one requested or just a piece of junk. By contrast, in a media streaming application, a user would quickly discover if the received information is good enough. The quality of service (QoS) metric used is also different in these two different applications. In a file-sharing application, the most important metrics are downloading time and the integrity of the received files. In a media streaming application, the more crucial performance parameters are the various playback quality metrics such as jitter, frame-rate, and resolution. Furthermore, the incentive techniques surveyed in this chapter subsume the underlying P2P network topology. Specifically, in most proposed systems, the communication message exchange mechanism is not explicitly modeled.

In a file-sharing system, users would like to retrieve files from other users, and would expect other users to do the same. Thus, each user would need to expend two different forms of resource:

- **Storage:** Each user has to set aside some storage space to keep files that may be needed by other users, even though such files may not be useful to the user itself;
- **Bandwidth:** Each user has to devote some of its outbound bandwidth for uploading requested files to other users;

Users usually perform file selection (and hence, peer selection) with the help of some directory system which may or may not be fully distributed. For example, in Napster [14], the directory is centralized.

There has been an increasing interest on P2P media streaming [64, 72, 77–79]. Indeed, in some practical systems, a single media server can support tens of thousands of concurrent P2P users [64, 77, 79], making it feasible to broadcast high-quality media content on the global scale without incurring prohibitively high infrastructure cost.

P2P media streaming is attractive due to its potentially high scalability. Indeed, it was observed that the performance of media streaming could increase with the peer population size [56]. Most P2P media streaming protocols organize peers in an unstructured overlay. Coolstreaming [77] is one of the pioneering protocols. Unlike traditional tree-based approaches [64, 72], it allows neighboring peers to exchange media packets in both directions, which provides better resilience to peer dynamics. Chainsaw [56] is a similar protocol where neighboring peers exchange media packet with one another by maintaining two sliding windows: window of interest and window of availability. GridMedia [79] improves the performance of Coolstreaming by actively pushing packets from one peer to another to reduce delay. Chunkyspread [68] addresses the issue of peers with heterogeneous bandwidth. Specifically, the stream of media packets is divided into a number of slices, which are distributed separately over different logical P2P overlays. Peers can then join different slices depending on their capabilities. mTreebone [70] is a combination of tree and unstructured approaches. It allows some static peers to form a tree backbone for shorter delay while other peers form an unstructured overlay for better resilience.

BitTorrent [30] is by far one of the most successful P2P file-sharing system. A key feature in BitTorrent is that each shared file is divided into pieces (of size 256 KB each), which are usually stored in multiple different peers. Thus, for any peer in need of a shared file, parallel downloading can take place in that the requesting peer can use multiple TCP connections to obtain different pieces of the file from several distinct peers. This feature is highly effective because the uploading burden is shared among multiple peers and the network can scale to a large size. Closely related to this parallel downloading mechanism is the incentive component used in BitTorrent. Specifically, each uploading peer selects up to four requesting peers in making uploading connections. The selection priority is based on descending order of downloading rates from the requesting peers. That is, the uploading peer selects four requesting peers that have the highest downloading rates. Here, downloading rate refers to the data rate that is used by a requesting peer in sending out pieces of some other file. Thus, the rationale of this scheme is to provide incentive for each participating peer to increase the data rate used in sending out file data (i.e.,

Table 1 A qualitative comparison of different P2P applications

Name	Type	Protocol	Architecture	Additional information
BitComet [1]	File sharing	BitTorrent	Structured	Closed source; adware
eMule [4]	File sharing	eDonkey2000, Kad	Decentralized	GPL open source; has a large user space
GreenTea [8]	Networked computing	Proprietary	Hybrid	Closed source; commercial product
KaZaA [12]	File sharing (music)	FastTrack	Hybrid	Closed source; adware/spyware
iMesh [10]	File sharing (music and video)	eDonkey2000, FastTrack, Gnutella	Hybrid	Closed source; freeware; supports social networks, purchase of copyrighted materials
Joost [11]	Multimedia streaming	P2PTV	Hybrid	Closed source; freeware; delivers near-TV resolution images; ad-supported service
Napster [14]	File sharing (music)	Napster	Centralized	Closed source; freeware; acquired by Roxio in 2003, providing paid music service without P2P technology
PPLive [15]	Multimedia streaming	P2PTV	Hybrid	Closed source; freeware; ad-supported service
Skype [16]	Voice-over-IP (VoIP)	KaZaA-alike	Hybrid	Closed source; freeware; offers paid service to initiate and receive calls via regular telephone numbers
Tribler [18]	File sharing (video)	BitTorrent	Structured	GPL open source; incorporates a keyword search protocol; supports social networks for content recommendation
WinMX [21]	File sharing (music)	OpenNap, proprietary WPNP	Centralized	Closed source; freeware; official WinMX central servers were shut down in 2005

uploading, or, in BitTorrent's term, *unchoking*). There are other related mechanisms (e.g., optimistic unchoking), which are described in detail in [30, 57].

Table 1 summarizes the features of several popular P2P applications.

3 Incentive Issues

To deter or avoid free-riding behaviors, the P2P community has to provide some *incentives*—returns for resource expenditure that are, more often than not, tangible and immediate. Such incentives would then motivate an otherwise selfish user to rationally choose to cooperate because such cooperation would bring tangible and immediate benefits. To mention an analogy, in human society, getting pay for our work is a tangible and immediate incentive to motivate us to devote our energy, which could otherwise be spent on other activities. Indeed, it is important for the incentive to be tangible so that a user can perform a cost–benefit analysis—if benefit outweighs cost, the user would then take a cooperative action [48]. It is also important for the incentive to be immediate (though this is a relative concept) because any resource is associated with an opportunity cost in that if immediate return cannot be obtained from a cooperative action, then the user might want to save the effort for some other private tasks.

3.1 Overview

To provide incentives in a P2P computing system, there are basically five different classes of techniques.

1. **Payment-Based Mechanisms:** Users taking cooperative actions (e.g., sharing their files voluntarily) would obtain payments in return. The payment may be real monetary units (in cash) or virtual (i.e., some tokens that can be redeemed for other services). Thus, two important components are needed: (1) currency and (2) accounting and clearing mechanism. Obviously, if the currency is in the form of real cash, there is a need for a centralized authority, in the form of an electronic bank, that is external to the P2P system. If the currency is in the form of virtual tokens, then it might be possible to have a P2P clearing mechanism. In both cases, the major objective of is to avoid fraud at the expense of significant overhead. Proper pricing of cooperative actions is also important—overpriced actions would make the system economically inefficient while underpriced actions would not be able to entice cooperation.
2. **Auction-Based Mechanisms:** In some situations, in order to come up with an optimal pricing, auctioning is an effective mechanism. In simple terms, auction involves bidding from the participating users so that the user with the highest bid get the opportunity to serve (or to be served, depending on context). An important issue in auction based systems is the valuation problem—how much a user should set in the bid? If every user sets a bid higher than its true cost in

providing a service, then the recipient of the service would pay too much than is deserved. On the other hand, if the bids are too low, the service providers may suffer. Fortunately, in some form of auctions, proper mechanisms can be constructed to induce bidders to bid at their true costs.

3. **Exchange-Based Mechanisms:** Compared to payment- and auction-based systems, exchange (or barter)-based techniques manifest as a purer P2P interaction. Specifically, in an exchange-based environment, a pair of users (or, sometimes, a circular list of users) serve each other in a rendezvous manner. That is, service is exchanged in a synchronous and stateless transaction. For example, a pair of users meet each other and exchange files. After the transaction, the two users can forget about each other in the sense that any future transaction between them is unaffected by the current transaction. This has an important advantage—very little overhead is involved. Most importantly, peers can interact with each other without the need of intervention or mediation by a centralized external entity (e.g., a bank). Furthermore, free riding is impractical. Of course, the downside is that service discovery and peer selection (according to price and/or quality of service) could be difficult.
4. **Reciprocity-Based Mechanisms:** While pure barter-based interactions are stateless, reciprocity generally refers to stateful- and history-based interactions. Specifically, a peer A may serve another peer B at time t_1 and does not get an immediate return. However, the transaction is recorded in some history database (centralized in some external entity or distributed in both A and B). At a later time $t_2 > t_1$, peer B serves peer A, possibly because peer B selects peer A as the client due to the earlier favor from A. That is, as peer A has served peer B before, peer B would give a higher preference to serve peer A. A critical problem is: how to tackle a special form of free-riding behavior, namely the “whitewashing” action (i.e., a user leaves the system and rejoins with a different identity), which enables the free rider to forget about his/her obligations.
5. **Reputation-Based Mechanisms:** A reputation-based mechanism is a generalized form of reciprocity. Specifically, while a reciprocity record is induced by a pair of peers (or a circular list of more than two peers), a reputation system records a score for each peer based on the assessments made by many peers. Each service provider (or consumer, depending on the application) can then consult the reputation system in order to judge whether it is worthwhile or safe to provide service to a particular client. Reputation-based mechanism is by nature globally accessible and thus, peer selection can be done easily. However, the reputation scores must be securely stored and computed, or otherwise, the scores cannot truly reflect the quality of peers. In some electronic market place such as eBay, the reputation scores are centrally administered. But such an arrangement would again need an external entity and some significant overhead. On the other hand, storing the scores in a distributed manner at the peers would induce problems of fraud. Finally, similar to reciprocity-based mechanisms, whitewashing is a low cost technique employed by selfish users to avoid being identified as a low-quality users which would be excluded from the system.

3.2 Payment Based Systems

Hauscheer et al. [42] suggested a token-based accounting system that is generic and can support different pricing schemes for charging peers in file sharing. The proposed system is depicted in Fig. 1. The system mandates that each user has a permanent ID authenticated by a certification authority. Each peer has a token account keeping track of the current amount of tokens, which are classified as local and foreign. A peer can spend its local tokens for accessing remote files. The file owner treats such tokens as foreign tokens, which cannot be spent but need to be exchanged with super-peers for new local tokens. Each token has a unique ID so that it cannot be spent multiple times.

At the beginning of each file-sharing transaction, the file consumer tells the file owner about which tokens it intends to spend. The file owner then checks against file consumer's account kept at the file owner's machine. If the tokens specified are valid (i.e., they have not been spent before), then the file consumer can send the tokens in an unsigned manner to the file owner. Upon receipt of these unsigned tokens, the file owner provides the requested files to the file consumer. When the files are successfully received, the file consumer sends the signed version of the tokens to the file owner. In this manner, Hauscheer et al. argued that there is no incentive for the peers to cheat.

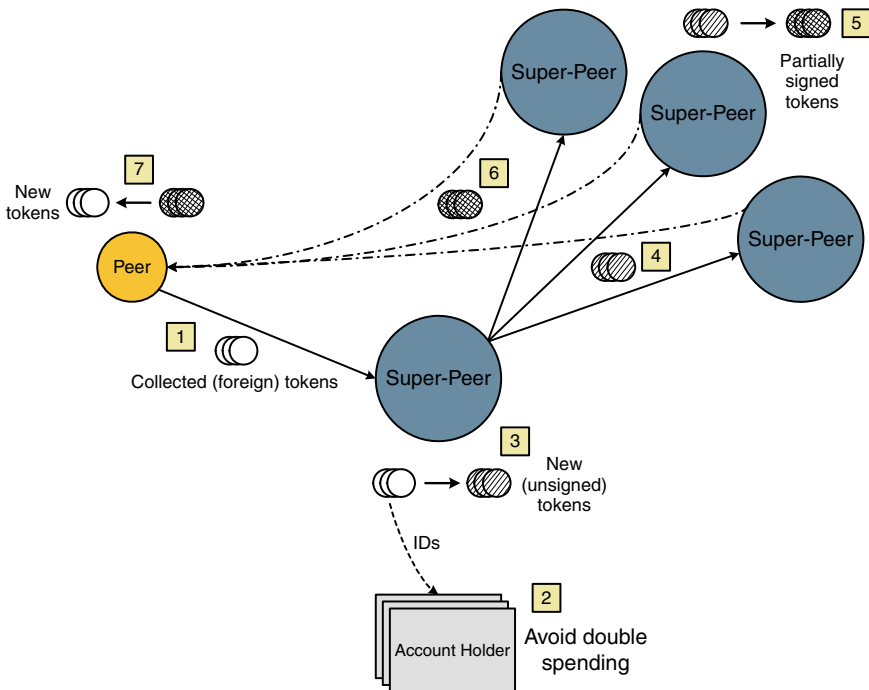


Fig. 1 A super-peer-based token accounting system for P2P file sharing [42]

Yang and Garcia-Molina [74] proposed the PPay micropayment system in which each peer can buy a coin from a broker. The peer then becomes the “owner” of the coin and can spend it to some other peer. An important feature is that even after the coin is spent, the original owner still has the responsibility to check the subsequent usage of the coin. For example, suppose A is the owner of a coin which is spent to B. If B wants to spend the coin in turn to C, the original owner A needs to check whether such a transaction is valid (e.g., to avoid double spending of the same coin). If A is off-line (e.g., temporarily departed the P2P system), then the broker is responsible to perform such checking.

Although the PPay system described above is a useful tool for supporting P2P sharing, Jia et al. [44] observed that PPay can be further improved. Specifically, Jia et al. proposed a new micropayment system, called CPay (an improved version of PPay), which has one significant new feature. The new feature is that the broker judiciously selects the most appropriate peer to be the owner of a coin. Specifically, the owner of a coin should be one that is expected to stay in the system for a long period of time. Thus, the broker’s potential burden of checking coin owners’ transactions can be considerably reduced.

3.3 Auction-Based Approaches

Gupta and Somani [41] proposed an auction-based pricing mechanism for P2P file object lookup services. In their model, each resource (e.g., a file object) is stored in a single node. However, the indices for such a file object are replicated at multiple nodes in the network and these nodes are called *terminal nodes*. When a peer initiates a lookup request for a certain file object, the request is sent through multiple paths toward the terminal nodes, as shown in Fig. 2. The problem here is that the intermediate nodes need some incentives in order to participate in the request forwarding process.

Gupta and Somani [41] suggested a novel solution to the incentive problem. Specifically, the initiating peer attaches a price in the request message it sends to the first layer of nodes in the request chains. Each intermediate node on the request chains then updates the price by adding its own “forwarding cost.” The terminal nodes also do the same updating before sending the request messages to the data source. Upon receiving all the request messages, the data source then performs a second price sealed bid auction (also referred to as Vickrey auction) [55] to select the highest bid among the terminal nodes. The selected terminal node then needs to pay the price equal to the value of the second highest bid. With this auction-based approach, all the intermediate nodes on the request chains have the incentive to participate in the forwarding process because they might eventually get paid by the requester should their respective request chains win the auction.

For example, consider the lookup process shown in Fig. 3. We can see that the request chain terminated by node T1 wins the auction process and the payoff to the data source node B is 60. The only intermediate node (node 1) then also gets a

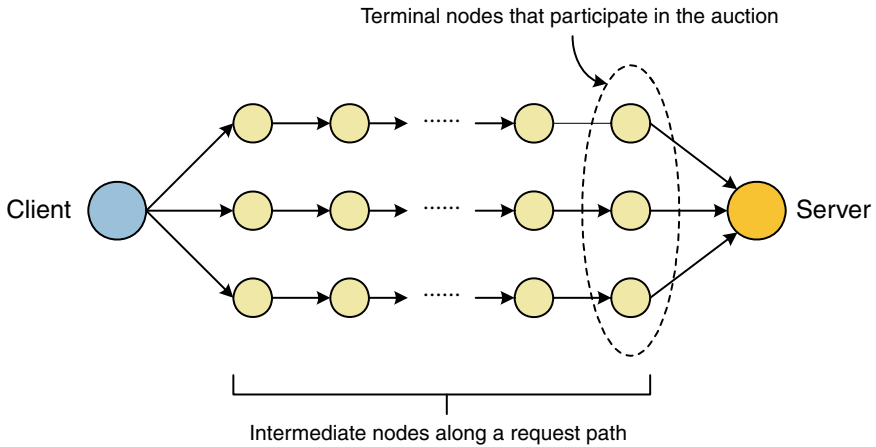


Fig. 2 The request forwarding process [41]

payoff. Gupta and Somani [41] also showed that a truthful valuation in is the optimal strategy for each intermediate node. Furthermore, based on the requirement that every message cannot be repudiated, it is also shown that the proposed mechanism can handle various potential threats such as malicious auctioneer, collusion between data source and a terminal node, and forwarding of bogus request message.

Wongrujira and Seneviratne [73] also proposed a similar auction-based charging scheme for forwarding nodes on a path from a requesting peer to a data source. However, they pointed out an important observation that some potential malicious peers could try to reduce the profits of other truthful peers by dropping the price messages. To mitigate this problem, a reputation system is introduced in that every peer maintains a history of interactions with other peers. The reputation value of a peer is increased every time a message is forwarded by such a peer. On the other

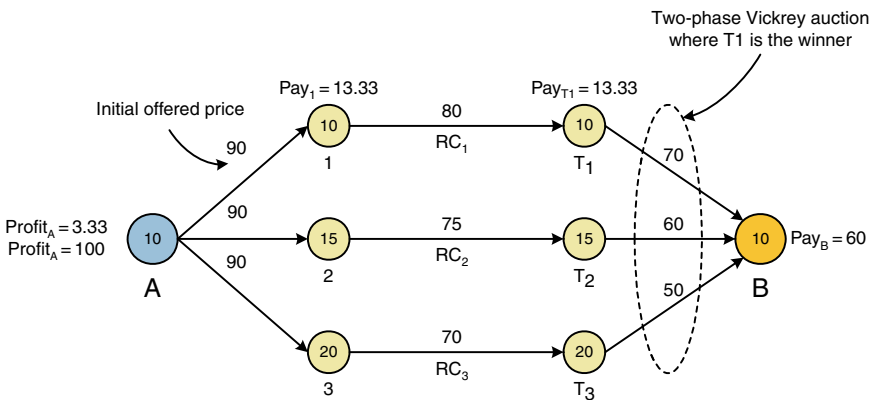


Fig. 3 An example of the auction process in request forwarding [41]

hand, if an expected message exhibits a timeout, the responsible peer's reputation value is decreased.

Wang and Li [69] also considered a similar problem in which a peer needs to decide how much to charge for forwarding data. Instead of using auction, a comprehensive utility function is used. The utility function captures many realistic factors: the quantitative benefits of forwarding data, the loss in delivering such data, the cost and the benefit to the whole community. With this utility function, an upstream peer has the incentive to contribute its forwarding bandwidth while a downstream peer is guided toward spending the upstream bandwidth economically. Furthermore, a reinforcement learning component is incorporated so that each peer can dynamically adjust the parameters in its utility function so as to optimally respond to the current market situations.

Sanghavi and Hajek [61] observed that in a typical auction based pricing mechanism as described above, there is a heavy communication burden on the peers. Indeed, the entire set of user preferences has to be communicated from a peer to the auctioneer. Sanghavi and Hajek then analytically derived a class of alternative information mechanisms that can significantly reduce the communication overhead. Specifically, each peer's bid is only a single real number in each case, instead of an entire real-valued function.

Hausheer and Stiller [43] studied a completely decentralized auction approach for electronic P2P pricing of goods in a system called PeerMart (which is built on top of Pastry [60]). The key idea is the usage of a broker set which comprises other peers in the electronic marketplace. Specifically, a broker set consists of peers whose IDs are closest to the ID of the good in the auction. Each of these peers then potentially acts as the auctioneer in the selling process. The advantage of the broker set based method is that in case a particular peer in the set is faulty (or even malicious in the sense that it does not respond to auction requests), another member in the set can take up the role of auctioneer. An example is shown in Fig. 4.

3.4 Exchange-Based Systems

Motivated by the fact that any payment/credit-based system entails a significant transaction and accounting overhead, Anagnostakis and Greenwald [22] proposed an exchange-based P2P file-sharing system. The fundamental premise is that any peer gives priority to exchange transfers. That is, in simple terms, any peer is willing to send a file to a peer that is able to return a desired file. However, based on this idea, it is incorrect to consider 2-way exchanges only. Indeed, a "ring" of exchange involving two or more peers, as shown in Fig. 5, is also a proper P2P file transfer.

In the exchange-based P2P file-sharing system, each peer maintains a data structure called *incoming request queue* (IRQ). Now, a crucial problem is how each peer can determine whether an incoming request should be entertained, i.e., whether such a request comes from some peer on a ring of exchange requests. It is obviously computationally formidable to determine all the potential multi-peer cycles. Fortunately,

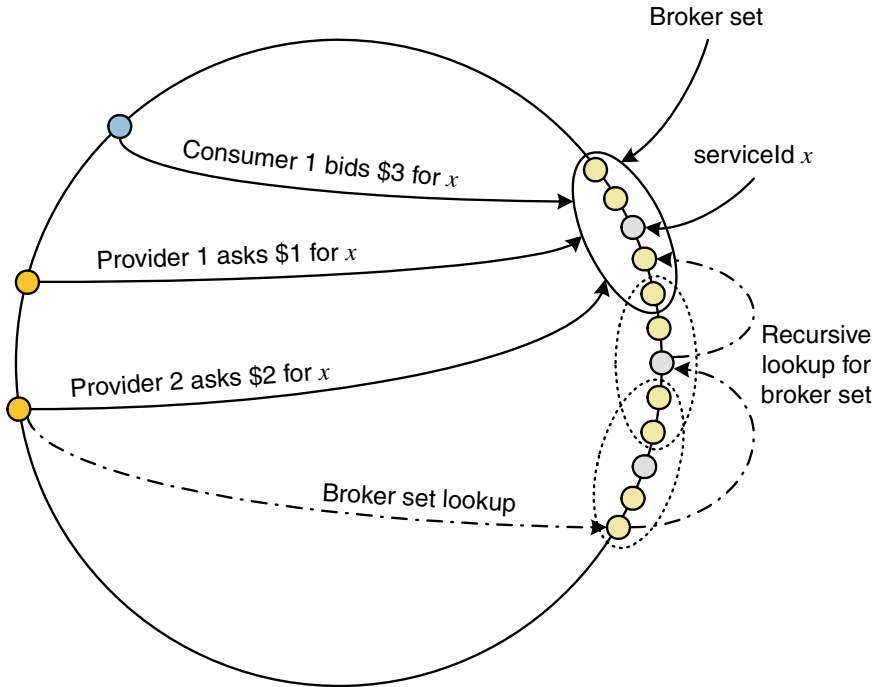


Fig. 4 An example of fully decentralized auction [43]

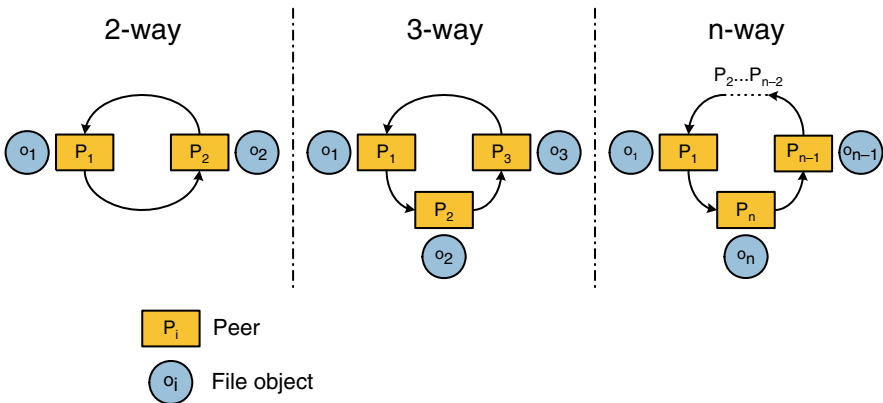


Fig. 5 Different feasible forms of exchanges [22]

Anagnostakis and Greenwald [22] argues that based on simulation results, in practice a peer only needs to check for cycles with up to five peers.

Each peer uses a data structure called *request tree* to check for potential request-cycles. For example, as we can see in Fig. 6, a peer A decides to entertain a request for file object o_2 because A finds that peer P_9 possesses an object that is needed by

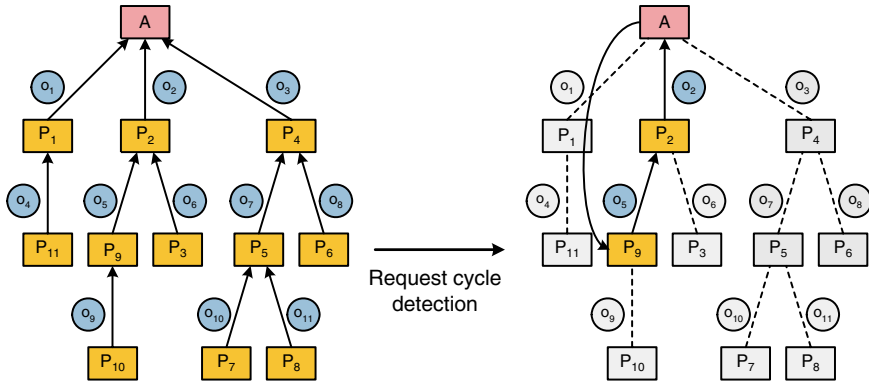


Fig. 6 Request cycle detection using the request tree data structure maintained at each peer [22]

A. Based on this checking mechanism, the incoming requests are prioritized. Simulation results indicate that the proposed exchange based mechanisms are effective in terms of file object download time.

3.5 Reciprocity and Reputation-Based Systems

Feldman et al. [37] suggested an integrated incentive mechanism for effectively deterring (or penalizing) free riders using a reciprocity-based approach. Specifically, the proposed integrated mechanism has three core components: discriminating server selection, maxflow-based subjective reputation computation, and adaptive stranger policies.

In the discriminating server selection component, each peer is assumed to have a private history of transactions with other peers. Thus, when a file-sharing request is initiated, the peer can select a server (i.e., a file owner) from the private history. However, in any practical P2P-sharing network, we can expect a high turnover rate of participation. That is, a peer may only be present in the system for a short time. Thus, when a request needs to be served, such a departed peer would not be able to help if it is selected. To mitigate this problem, a shared history is to be implemented. That is, each peer is able to select a server from a list of global transactions (i.e., not just restricted to those involved the current requesting peer). A practical method of implementing shared history is to use a distributed hash table (DHT)-based overlay networking storage system [65]. Specifically, a DHT is an effective data structure to support fast lookup of data locations.

A problem in turn induced by the shared history facility is that collusion among non-cooperative users may take place. Specifically, the non-cooperative users may give each other a high reputation value (e.g., possibly by reporting bogus prior transaction records). To tackle this problem, Feldman et al. suggested a graph theoretic technique. To illustrate, consider the reputation graph shown in Fig. 7. Here, each node in the graph represents a peer (*C* denotes a colluder) and each directed edge

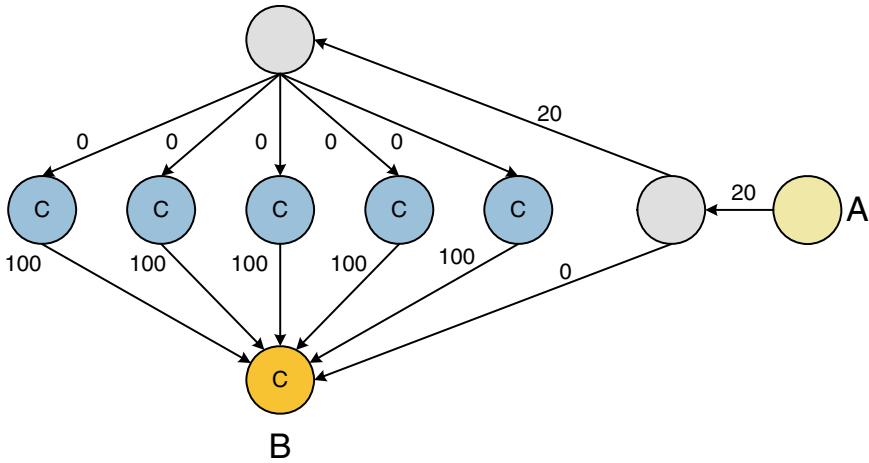


Fig. 7 A graph depicting the perceived reputation values among peers (C denote a colluder) [37]

represents the perceived reputation value (i.e., the reputation value of the node incident by the edge as perceived by the node originating the edge). We can see that the colluders give each other a high reputation values. On the other hand, a contributing peer (e.g., the top node) gives a reputation value of 0 to each colluder because the contributing peer does not have any prior successful transaction carried out with a colluder. With this graph, we can apply the maxflow algorithm to compute the reputation value of a destination peer as perceived by a source peer. For instance, peer B's (the destination) perceived reputation value with respect to peer A (the source) is 0 despite that many colluders give a high reputation value to B.

Finally, an adaptive stranger policy is proposed to deal with whitewashing. Instead of always penalizing a new user (which would discourage expansion of the P2P network), the proposed policy requires that each existing peer, before deciding whether to do a sharing transaction with a new user, computes a ratio of amount of services provided to amount of services consumed by a new user. If this ratio is great than or equal to 1, then the existing peer will work with the new user. On the other hand, if the ratio is smaller than 1, then the ratio is treated as a probability of working with this new user.

Sun and Garcia-Molina [66] suggested an incentive system called Selfish Link-based InCente (SLIC), which is based on pairwise reputation values. Specifically, any peer u maintains a reputation value $W(u, v)$ for each of its neighbor peer v , where the reputation value is normalized such that $0 \leq W(u, v) \leq 1$. Here, "neighbor" means a peer v currently having a logical connection with u and thus, such a peer v can potentially request for service from u . With these reputation values, the peer u can then allocate the uploading bandwidth to any requesting neighbor peer v with a value of $W(u, v) / \sum_i W(u, i)$. The reputation value $W(u, v)$ is updated periodically based on an exponential averaging method.

Under this model, Sun and Garcia-Molina [66] observed that each peer has the incentive to do some or all of the following, in order to increase its reputation values as perceived by other peers (and hence, enjoy a better quality of service):

- Sharing out more file data;
- Connecting to more peers (to increase the opportunities for serving others);
- Increasing its total uploading capacity.

3.6 Penalty-Based Approaches

Feldman et al. [35] also investigated disincentive mechanisms that can discourage free riding. Specifically, they considered various possible penalty schemes in deterring free riders. A simple model is used. At the core of the model, each user i in the P2P-sharing network is characterized by a positive real-valued *type* variable, denoted as t_i . Another key feature of the model is that the cost of contributing is equal to the reciprocal of the current percentage of contributors, which is denoted as x . Thus, for any rational user with type t_i , the user will choose to contribute if $1/x < t_i$ and free ride if $1/x \geq t_i$.

Furthermore, the benefit each user derived from the P2P network is assumed to be of the form αx^β , where $\beta \leq 1$ and $\alpha > 0$. With this benefit function, the system performance is defined as the difference between the average benefit and the average contribution cost. Specifically, system performance is equal to $\alpha x^\beta - 1$.

Even with the simplistic model described above, Feldman et al. provided several interesting conclusions. Firstly, it is found that excluding low type users can improve system performance only if the average type is low and α is large enough. Unfortunately, exclusion is impractical because a user's type is private and thus, cannot be determined accurately by other peers. It is then assumed that free-riding behaviors are observable (i.e., free riders can be identified). Such free-riders are then subject to a reduction in quality of service. Quantitatively, the benefit received by a free rider is reduced by a factor of $(1 - p)$, where $0 < p \leq 1$. A simple implementation of this penalty is to exclude a free rider with a probability of p . The second interesting conclusion is that the penalty mechanism is effective in deterring free riders when the penalty is higher than the contribution cost. In quantitative terms, the condition is that $p > 1/\alpha$. Finally, another interesting conclusion is that for a sufficiently heavy penalty, no social cost is incurred because every user will contribute (i.e., choose not to be a free rider) so that optimal system performance is achieved. In particular, to deal with the whitewashing problem, the analysis suggests that every new user is imposed a fixed penalty. Essentially, this is similar to the case in the eBay system where every new user has a zero reputation and thus, will less likely be selected by other users in commercial transactions. However, this is in sharp contrast to the adaptive stranger policies suggested also by Feldman et al. in another study [37] that we have described earlier.

3.7 Game Theoretic Modeling

Ranganathan et al. [58] proposed and evaluated three schemes induced by the Multi-Person Prisoner's Dilemma (MPD) [55, 62]. The basic Prisoner's Dilemma game models the situation where two competitors are both better off if they cooperate than when they do not. However, without communication, the unfortunate stable state is that both competitors would choose not to cooperate. An MPD is a generalization of the basic PD. Specifically, the key features of the MPD framework can be briefly summarized as follows:

- The MPD game is symmetric in that each of n players has the same actions, payoffs, and preferences.
- Any player's payoff is higher if other players choose some particular actions (e.g., "quiet" instead of "fink").

The MPD framework is used for modeling P2P file sharing as follows. There are n users in the system, each of which has a distinct file that can be either shared or kept only to the owner. The system is homogeneous in that all files have the same size and same degree of popularity. Now, the potential benefit gained by each user is the access of other users' files. The cost involved is the bandwidth used for serving other users' requests. With this simple model, it can be shown that the system has a unique Nash equilibrium in which no user wants to share. Obviously, this equilibrium is sub-optimal (both at the individual level and at a systemwide level) in that each user could obtain a higher payoff (i.e., a higher value of net benefit) if all users choose to share their files.

Motivated by the MPD modeling, Ranganathan et al. proposed three incentive schemes:

- **Token Exchange:** This is a payment-based scheme because each file consumer has to give a token to the file owner in the sharing process. Each user is given the same number of tokens initially and each file has the same fixed price.
- **Peer-Approved:** This is a reputation-based scheme in that each user is associated with a rating which is computed using metrics such as the number of requests successfully served by the user. A user can download files from any owner who has a lower or the same rating. Thus, to gain access to more files in the system, a user has to actively provide service to other users so as to increase the rating.
- **Service Quality:** This is also a reputation-based scheme similar to Peer-Approved. The major difference is that a file owner provides differentiated service qualities to users with different ratings.

Theoretical analysis [58] indicates that the Peer-Approved policy with a logarithmic benefit function (in terms of number of accessible files) can lead to the optimal equilibrium where every user contributes fully to the system. Simulation results also suggest that Peer-Approved generates performance (in terms of total number of files shared) comparable to that of Token Exchange, which entails a higher difficulty in practical implementation as it requires a payment system.

Becker and Clement [24] also suggested an interesting analysis of the sharing behaviors using variants of the classical 2-player Prisoner’s Dilemma. Specifically, the P2P file-sharing process is divided into three different stages: introduction, growth, and settlement. In the introduction stage, the P2P network usually consists of just a few altruistic users who are eager to make the network viable. Thus, sharing of files is a trusted social norm. The payoffs of the two possible actions (supply files or not supply files) are depicted in Fig. 8. Here, we have the payoffs ranking as: $R > T > S > P$ (note: T: Temptation, R: Reward, S: Sucker, P: Punishment). Consequently, the Nash equilibrium profile is: (Supply, Supply). Note that the payoffs ranking in the original Prisoner’s Dilemma is $T > R > P > S$, and as such, the Nash equilibrium is the action profile in the lower right corner of the table.

In the growth stage, we can expect that more and more non-cooperative users join the network. For these users, the payoffs ranking becomes: $T > R > P > S$, which is the same as the original Prisoner’s Dilemma. Thus, the Nash equilibrium for such users occurs at the profile: (No Supply, No Supply). As the P2P network progresses to the mature stage (i.e., the size of the network becomes stabilized), we can expect that a majority of users are neither fully altruistic nor fully non-cooperative. For these users, the payoffs ranking is: $R > T > P > S$. As a result, the payoff matrix is depicted in Fig. 9. As can be seen, there are two equally probable Nash equilibria: (Supply, Supply) and (No Supply, No Supply). Consequently, whether or not the P2P network is viable or efficient depends on the relative proportions of users in these two equilibria. Results obtained in empirical studies [24] using real P2P networks conform quite well to the simple analysis described above.

Ma et al. [52, 53] suggested an analytically sound incentive mechanism based on a fair bandwidth allocation algorithm. Indeed, the key idea is to model the P2P sharing as a bandwidth allocation problem. Specifically, the model is shown in Fig. 10. Here, multiple file requesting peers compete for uploading bandwidth of a source peer. Each requesting peer i sends a bidding message b_i to the source peer N_S . The source peer then divides its total uploading bandwidth W_S into portions of

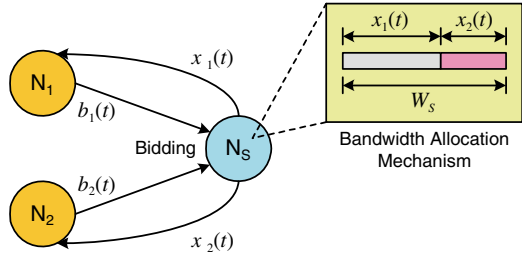
Fig. 8 Payoff table in the introduction stage [24]

		Player 2	
		Supply g_2^1	No Supply g_2^2
Player 1	Supply g_1^1	R R	S T
	No Supply g_1^2	T S	P P

Fig. 9 Payoff table in the settlement stage [24]

		Player 2	
		Supply g_2^1	No Supply g_2^2
Player 1	Supply g_1^1	R R	S T
	No Supply g_1^2	T S	P P

Fig. 10 Two file requesting peers (N_1 and N_2) compete for uploading bandwidth of a source peer (N_S) [52]



x_i for the peers. However, due to network problems such as congestion, each peer i may receive an actual uploading bandwidth of x'_i which is smaller than x_i .

Each bidding message b_i is the requested amount of bandwidth. Thus, we have $x_i \leq b_i$. To achieve a fair allocation, the source peer uses the contribution level C_i of each competing peer i to determine an appropriate value of x_i . Ma et al. [52, 53] described several allocation algorithms with different complexities and considerations: simplistic equal sharing, max–min fair allocation, incentive-based max–min fair allocation, utility-based max–min fair allocation, and incentive with utility-based max–min fair allocation. The last algorithm is the most comprehensive and effective. It works by solving the following optimization problem:

$$\max \sum_{i=1}^N C_i \log\left(\frac{x_i}{b_i} + 1\right) \tag{1}$$

where

$$\sum_{i=1}^N x_i \leq W_S \tag{2}$$

Here, the logarithmic function represents the utility as perceived by each peer i .

The above optimization problem can be solved by a progressive filling algorithm that prioritizes competing peers in descending order of the marginal utility $C_i/(b_i + x_i)$.

Given values of b_i and C_i , the source peer can compute the allocations in a deterministic manner. However, from the perspective of a requesting peer, a problem remains as to how it should set its bidding value b_i . Using a game theoretic analysis, it is shown that the action profile in which:

$$b_i = \frac{W_S C_i}{\sum_{j=1}^N C_j} \forall i$$

is a Nash equilibrium. Furthermore, provided that all cooperative peers use their respective strategies as specified in the Nash equilibrium action profile, collusion among non-cooperative peers can be eliminated. Note that each requesting peer i needs to know the values of W_S and $\sum_{j=1}^N C_j$ in order to determine its own bid b_i .

In a practical situation, these two values can be supplied by the source peer to every requesting peer.

4 Security Issues

Given the open and autonomic nature of a P2P system, it is very difficult, if not impossible, to completely avoid having malicious peers participating in the network. Consequently, unlike many traditional systems, a P2P system has to tolerate many different attacks while still provides useful services. The bottomline is that the P2P system can still operate with a sufficiently large population of peers that are interested in participating, taking up the risk of being attacked. In this section, we describe the basic mechanisms of various common attacks and the essence of recently proposed counter-measures.

4.1 Overview

There are many security problems in an open distributed system. Yet a P2P network has its unique challenges due to its fully distributed and dynamic operating characteristics [47, 50]. Indeed, there are several potentially detrimental attacks that can damage a P2P system:

- **Churn Attacks.** This is a P2P version of denial-of-service attacks in that malicious peers deliberately join and leave the system frequently, inducing a large amount of processing and communication overheads for population maintenance.
- **Poisoning and Pollution.** Some malicious peers intentionally injects falsified data (e.g., unusable files) into the shared pool of contents.
- **Sybil Attacks.** Because many P2P systems rely on voting to make collective decisions, it is important that peer IDs cannot be easily forged; otherwise, a peer can control a large portion of peer IDs and out-vote other honest peers.
- **Worms.** In many P2P file-sharing systems, peers download files without much checking and thus, such a platform is an idea environment for spreading worms.
- **Malwares.** Related to poisoning and worms, honest peers could inadvertently downloaded files encapsulating malwares such as virus codes.
- **Cheating in a P2P MMOG.** Many massively multi-player online games (MMOGs) are moving toward a P2P architecture from a traditional centralized client-server structure. However, such a move also gives away the centralized control over peers' actions and the associated effective security measures. In effect, as the game state updating and communication tasks are handled by the peers themselves, some malicious peers can cheat by launching many of the above attacks.

4.2 Churn Attacks

“Churning” refers to the situation where peers join and leave frequently so that the system population becomes highly varying over a short period of time. This high turnover of peers can easily increase the overheads in the P2P system in the sense that much computation and communication loads are devoted to handling the joining and leaving events, but not in the actual sharing operations. Effectively, a denial-of-service attack results in an unintentional manner.

Linga et al. [51] examined the efficacy of a probabilistic churn-resistant scheme called Kelips [40], which is based on a self-regenerating data structure. The basic design rationale of Kelips is a precursor to that of BitTorrent. Specifically, a Kelips system with n peers is partitioned into \sqrt{n} sets called *affinity groups*. Each peer then belongs to one affinity group chosen according to a consistent hashing function to map the peer’s ID (i.e., IP address and port number) into the set of group IDs i , where $i = 0, \dots, \sqrt{n} - 1$. Thus, unlike structured P2P topology such as Pastry, Tapestry, and Chord, Kelips allows a peer to pick any peer within its affinity group according to system features such as topology, trust, security concerns, etc. The latency of communications among peers within an affinity group ranges from $O(\log^2(n))$ to $O(\log(n))$. The soft state kept in each peer occupies only a small amount of memory. For instance, according to [51], in a system with 10 million files and 100,000 peers, the soft state kept in each peer requires only 1.93 MB of storage space. Experimental results using a real PC cluster indicated that the Kelips is effective in handling churn attacks.

4.3 Poisoning and Pollution

“Poisoning” is a type of attacks that undermines the integrity of the P2P system by inserting false information and/or identities of malicious peers into the sharing process.

Daswani and Garcia-Molina [34] studied a poisoning problem in a P2P protocol designed for Gnutella called GUESS (Gnutella UDP Extension for Scalable Searches) [33]. Based on random walks in unstructured P2P networks, the GUESS protocol was designed to tackle the problem of high overhead in the original Gnutella’s flooding approach in propagating queries. Specifically, according to the GUESS protocol, each peer keeps a cache of other peers that are available to accept queries, and sends its queries to one of the peers in its cache in a random fashion. To maintain the correctness in the query process, peers must delete from their respective caches the peers that are no longer available. To achieve this, peers need to exchange “ping” and “pong” messages (elaborated below), and thus, the caches are called “pong caches.”

Although the usage of pong caches can successfully mitigate the communication overhead problem, a new security problem arises because the pong caches can get poisoned by malicious peers. Specifically, if a malicious peer’s ID is stored in a pong

cache of a good peer, the latter's pong cache is considered as poisoned because it becomes very difficult for the latter to find other good peers for getting useful shared files.

Before describing Daswani and Garcia-Molina's proposed techniques for countering poison attacks, let us have a brief overview of the ping-pong mechanism in using the caches. In simple terms, each peer's cache contains a set of peer IDs that it believes they are still available in the system for obtaining files. In a totally asynchronous manner, each peer may ping another peer in its cache. A ping message is piggybacked on a file request. If the peer receiving the ping message is active, it responds with a pong message. While a ping message is a simple dummy packet, a pong message encapsulates a set of peer IDs that the responding peer possesses. Usually, the responding peer only includes a subset of its peer IDs in the pong message. Upon receiving the pong message, the requesting peer may replace a subset of its peer IDs from the cache, in a random manner, with some of the peer IDs contained in the pong message. Such randomized replacement is designed for a robust maintenance of valid cache entries.

However, if the pong cache of a peer N_i is poisoned, another requesting peer N_j might get the ID of a malicious peer N_k . Subsequently, N_j might send a ping to N_k , which will return a set full of malicious peer IDs in the pong message. Consequently, such poisons will propagate quickly in the network. Malicious peers, by nature, will not supply authentic file data. Thus, good peers will be unable to get useful information from the network and drop out. The system will then collapse.

To mitigate or more specifically "contain" the malicious effects of poisoning, Daswani and Garcia-Molina [34] suggested two simple algorithms. The first algorithm is called ID smearing algorithms (IDSA), which works by dropping IDs that are repeatedly received from pong messages. For example, if the ID N_2 already appears in N_1 's pong cache, then N_1 will delete N_2 from its pong cache upon getting N_2 again in some pong messages. The rationale is that such IDs are more likely from malicious peers. The second algorithm is called dynamic network partitioning (DNP). Each peer divides the peer ID space into a number of partitions and select only one of those partitions as an active partition. A peer only accepts peer IDs from its active partition for replacement purposes. By dynamically changing its active partition, a good peer can potentially avoid getting malicious peer IDs that could be clustered in a certain region in the ID space.

Simulation results indicated that the IDSA scheme can effectively limit poisoning when the number of malicious peers in the system is smaller than or approximately equal to the pong cache size. The DNP algorithm can effectively reduce the number of malicious peer IDs that can poison a pong cache.

Christin et al. [28] performed a detailed experimental study, based on a large-scale PlanetLab-based platform, on the problem of poisoning shared contents (instead of meta-data such as peer IDs) in the system. Specifically, they considered a popular content poisoning technique which works by randomly injecting a large number of decoys into the network. A related technique, which is called pollution, is to randomly inject a large number of unusable files into the network. Experimental

results indicate that the random poisoning techniques can have great impact on the perceived availability of desired files.

Recently, Kumar et al. [49] presented analytical fluid models of content poisoning propagation dynamics. Their fluid models generate a set of non-linear differential equations, from which closed-form solutions are derived. Their models are useful in that they capture a wide range of user behaviors including propensity for popular versions of files, abandonment after repeated failures, free riding, and local version blacklisting.

4.4 Sybil Attacks

Sybil attacks refer to the situations where a single malicious peer uses multiple, possibly authentic, identities to instantiate multiple virtual peers in the system. The major motivation is to control a relatively large share of population in the system, thereby winning any voting actions within the P2P network. Yu et al. [76] proposed a novel protocol called SybilGuard for limiting the adverse effects of sybil attacks. Specifically, SybilGuard works by establishing a “social network” among honest peers. That is, between each pair of honest peers, an edge is established if the human users represented by the two peers trust each other. Thus, such a social network relies on some off-line or out-of-band information to establish.

Once the social network among honest peers has been set up, conceptually the P2P system can be visualized as shown in Fig. 11. As can be seen, by nature of the social network, there will be very few, if any, edges between the set of honest users and the malicious sybil attackers. This is because although the sybil attackers can possibly obtain a large number of legitimate IDs, they cannot easily gain “trust” from the users of the honest peers. Effectively, the social network among honest peers effectively segregate themselves from the malicious peers.

Simulation results indicated that SybilGuard effectively guarantees that (1) the number and size of sybil groups are properly bounded for 99.8% of the honest users, and (2) an honest peer can accept, and be accepted by, 99.8% of all other honest peers.

Friedman [38] also designed a similar social-network-based system called “Good Neighbors” for propagating good worms (a topic of the next subsection) to patch software vulnerabilities.

Cerri et al. [26] examined a Sybil attack scenario in a structured P2P system. Specifically, they considered the DHT-based P2P system called Kademlia [54], in which each peer is identified by a unique 160-bit string. Most importantly, each peer can choose its bit string freely using a random function. The premise is that collisions in the ID space are statistically improbable, given that the ID space has 2^{160} distinct elements. A key feature of Kademlia is that a peer is responsible for shared resources with indices near to the peer’s ID. Distance between identifiers (also, between peers and resources) is measured using an XOR operation—performing a bit-wise XOR between the peer’s ID and the index. Given this property and

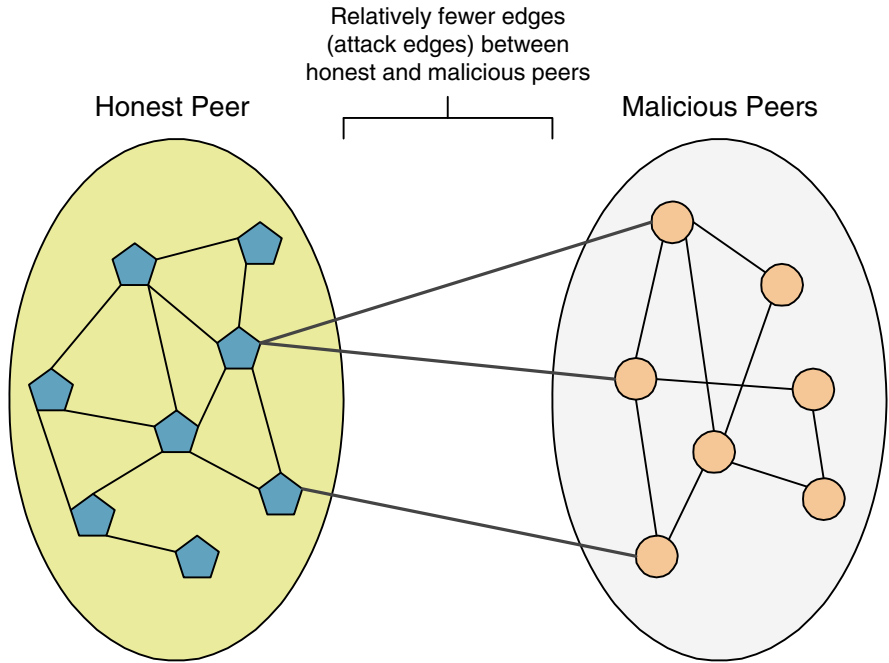


Fig. 11 The social network among honest peers effectively segregate themselves from the malicious peers [76]

the sparsely populated ID space, conceptually an attacker can impersonate a peer (hence, provide bogus resources subsequently) by getting an ID close to that of a legitimate peer. In practice, each resource is replicated at k peers which are those k peers with IDs closest to the resource ID. Thus, an attacker has to obtain k IDs in order to completely shield the resource.

Cerri et al. examined three different types of attacks: complete attacks, dictionary attacks, and partial attacks. In a complete attack, the attacker must obtain k valid IDs which are closer than any other ID to the target resource ID. It is found [26] that the complexity of this attack depends only on the population size of the system, and grows linearly with the factor k . A dictionary attack refers to the situation where the attacker stores all the randomly obtained IDs associated with each targeted resource ID to build a table. After such a dictionary is built, the attacker can then sample the whole ID space by picking values from the dictionary. It is envisioned that the computation time and storage space required for doing so are within the capabilities of even a desktop computer system. In a partial attack, the attacker wants to control only a fraction c of the k IDs that associate with the targeted resource. Cerri et al. then suggested two simple solutions to tackle these attacks. The first solution is to constrain the computation of an ID by enforcing the use of connection state information such as IP address and port number of the peer. Obviously, this solution can allow easier verification of the peer's identity. However, a drawback is that the

ID could be changed from time to time due to departure and re-joining the system. The second solution is based on a more stable state information by using a hash of the public key possessed by the peer. How these two IDs can be combined remains a challenging research problem.

4.5 Worms

Chen and Gray [27] studied the characteristics of three different types of non-scanning worms which are highly detrimental to a P2P system. Indeed, as observed by Chen and Gray, in a highly popular P2P file-sharing system (e.g., Gnutella and BitTorrent), users may inadvertently download decoy files embedded with malicious code or malware (the topic in the next subsection). More importantly, the P2P client software may contain vulnerabilities that could be exploited by attackers. This is aggravated by the fact that usually a large fraction of P2P users run the exact same software. Consequently, exploiting one single vulnerability can already potentially damage a large fraction of the P2P system. An important implication is that attackers do not really need to use scanning worms which can be detected by intrusion detection systems (IDSs) through monitoring scanning activities. Rather, attackers now increasingly employ non-scanning worms, which make use of legitimate networking actions (e.g., ping and pong messages in Gnutella) to propagate.

Chen and Gray considered three types of non-scanning worms:

- Passive worms that are embedded in malwares, awaiting users to execute them;
- Reactive worms that propagate and replicate through legitimate network actions;
- Proactive worms that judiciously infect other peers by using topological information.

Chen and Gray then presented a unified simulation framework modeling an unstructured file-sharing P2P network. Based on a detailed simulation study, Chen and Gray found that passive worms usually do not cause major outbreaks because they spread rather slowly. Once immunization is applied, the spreading of the worms quickly die out. In contrast, reactive worms are stealthy and can infect a much large fraction of peers in a relative short period of time compared to the case of passive worms. The proactive worms are the most detrimental in that they intelligently make use of valid peer addresses (e.g., in a pong cache) for spreading. Fortunately, similar to a scanning worm, a proactive worm usually generates noticeable amount of network traffic anomalies (e.g., connection failures due to peers' departures) and, as such, could be detected by an anomaly-based IDS.

Yu et al. [75] proposed a region-based active immunization defense strategy to combat proactive worms. In their proposed approach, some peers are designated to perform the task of defending against proactive worms. Such peers are further classified into two categories: ordinary defense peer and defense region leader. The responsibilities of the former include local worm detection, reporting anomalies to the region leader, and carrying out defense commands from the region leader.

Accordingly the defense region leader is a centralized agent for determining whether a worm propagation is identified based on the information gathered from the ordinary defense peers in the region. The rationale behind the proposed region-based defense system is inspired by the adaptive immune systems where white blood cells (called lymphocytes) cooperating with each other to track down intruding pathogens. Such pathogens are then eliminated by the active immunization process.

Specifically, the worm detection component in each ordinary defense peer judiciously analyzes the incoming and outgoing traffic. Detected traffic anomalies are then reported to the region leader, which carries out data fusion and correlation analysis. Some decision rules, such as “at least K defense hosts report anomalies,” can then be applied. Once a worm propagation is confirmed by the region leader, counter-worm immunization is then carried out by the defense peers. Their mathematical analysis and simulation results indicated that system parameters such as defense list size, worm detection success ratio, and immunization rate have the most crucial impacts on the overall defense performance.

4.6 Malwares

Kalafut et al. [46] conducted a detailed experimental study on the prevalence of malware in P2P networks. They considered two different open source P2P networks, Limewire [13] and OpenFT [6]. Their experimental data based on over 1 month of data indicate that 68% of all downloadable responses in Limewire containing archives and executables encapsulating malware. For OpenFT, the corresponding value is 3%. Furthermore, most infections are from a very small number of distinct malware. Specifically, in Limewire, the top three most prevalent malware account for 99% of all the malicious responses. For OpenFT, the corresponding value is 75%. Kalafut et al. also investigated the sources of malicious responses. It is found that 28% of all malicious responses in Limewire come from private address ranges. For OpenFT, one single host contributes 67% of all malicious responses. Based on these useful findings, Kalafut et al. suggested a simple method for filtering malware. Specifically, the method is to filter downloads based on the most commonly seen sizes of the most popular malware. Doing such filtering is found to be able to block a large portion of malicious files with a very low rate of false positives. Indeed, compared to the Limewire’s mechanism of malware detection which can only track 6% of malicious responses, their proposed size-based filtering method can detect over 99% of them.

Shin et al. [63] also conducted a detailed study on the prevalence of malware downloading in the KaZaA file-sharing system. With a lightweight crawler built for the KaZaA file-sharing network, they gathered data about more than 500,000 files returned by the KaZaA network in response to 24 common query strings. Shin et al. used 71 different malicious programs (e.g., viruses, Trojan horses) to construct 364 different signatures. Their major finding is that over 12% of KaZaA client hosts are infected by more than 40 different virus programs. Furthermore, over 20% of the total crawled data are virus-containing files.

4.7 Cheating in P2P MMOG

Massively, MMOGs are becoming one of the major Internet activities in a global scale. For a typical MMOG, it is not uncommon to have tens of thousands to hundreds of thousands of players, that are possibly widely dispersed geographically, actively involved in the game at the same time. In such a large scale, given the intensive computations (e.g., game state updating, high-resolution graphics rendering based on the updates) and communications (e.g., game state updates, commands and control) required, the original centralized approach becomes prohibitively expensive. Furthermore, the centralized server (or server farm) is an obvious single point of failure. Thus, to improve scalability, availability, and performance, recently MMOGs have been migrated to a P2P architecture in the sense that the players (i.e., the peers) take up the responsibilities of updating the game state and the communications involved. As a case in point, the current leader of MMOG, World of Warcraft [20], has already moved the distribution of client software updates to a P2P architecture. It is believed that the game itself will also be executed in a large part on a P2P network.

However, using a P2P architecture which is usually unstructured, poses a new security problem. Specifically, while a centralized server manifests itself as an authority trusted by all players, P2P communications inevitably untrustworthy. Indeed, a participating peer might be malicious and send false updates to other peers. An even worse situation is that a set of malicious peers form a collusion so as to gain a collective advantage (e.g., by delaying useful information to other honest players). In general, there are five practicable cheating actions [39, 45, 71]:

1. **Fixed-Delay Cheat.** The reception of state updates to a malicious peer is faster than the transmission from it. Thus, such a malicious peer can gain a timing advantage over the opponents.
2. **Time-Stamp Cheat.** A malicious peer puts bogus time stamps—earlier rather than later—on state update messages to gain timing advantage over the opponents.
3. **Suppressed Update Cheat.** A malicious peer withholds some important state updates from reaching the opponents so that the latter could make erroneous decisions.
4. **Inconsistency Cheat.** A malicious peer sends different bogus update messages to different opponents to confuse them.
5. **Collusion Cheat.** A set of malicious peers form a collusion to launch larger scale cheating, and more importantly, to protect against detection (elaborated below).

To tackle these threats in a P2P MMOG, GauthierDickey et al. [39] proposed a protocol called New Event Ordering (NEO), which is a commit and reveal algorithm with majority agreement on valid commitments. Specifically, in a P2P MMOG with NEO, computations and communications are synchronized in fixed length rounds. At the end of each round, each participating peer executes a voting function so as to reach a consensus over the updated game state. While the NEO protocol is simple

to implement and hence highly efficient, Corman et al. [31] later discovered that it cannot defend against the following cheating actions:

- A malicious peer can replay obsolete updates to some opponents.
- A malicious peer can construct update messages that are valid in the sense that they are associated with signed votes.
- A malicious peer can still send different update messages to different opponents.

The first two problems are due to the fact that in NEO, update components in a message are simply concatenated but not integrated in a cryptographically secure manner. The third problem is actually a result of the first—a malicious peer can replay old updates from previous rounds to some opponents. Corman et al. [31] proposed a new protocol called Secure Event Agreement (SEA) which is designed to handle the above problems. The SEA protocol signs the whole game state update message and integrates all components in a cryptographically secure method. The SEA protocol also makes use of additional system information items such as session ID, a nonce that is unique for each round, and the round number.

4.8 Other Security Techniques

There are many novel P2P security techniques proposed recently [67]. Due to space limitations, we briefly mention two particularly interesting ones.

Danezis and Anderson [32] presented an interesting theoretical study on the economics of resisting censorship of P2P shared contents. Using a simple unimodal utility function to capture each peer's preference over shared contents, they analyzed the costs of imposing and resisting censorship in the system, by some malicious external entity and the participating peers, respectively. Their major insight is that a highly heterogeneous population of peers in terms of preferences can better withstand censorship attacks.

Capkun et al. [25] reported an insightful study on the effect of mobility on the security of a wireless P2P system. Assuming that each mobile device has a secure side channel for communicating with other devices, they showed that the temporary vicinity of peers can actually benefit the set up of security associations between users who were previously separated wide apart from each other.

5 Discussion and Future Work

There are several promising avenues of further research in incentive mechanisms. Firstly, much work should be done in improving the accountability and resilience of tangible incentive systems (such as payment based or exchange based). Specifically, robust schemes should be designed to guard against forging of benefits (e.g., payment tokens) and avoiding “inflation” caused by resource-rich peers overwhelming resource-poor peers (e.g., peers that just join the network). Secondly, while

many game theoretic schemes have been suggested, most of them are based on complete information. Obviously, in a practical P2P system, each peer can possibly know the information about its immediate neighbors only. Even then, such knowledge can be very inaccurate. Thus, there is a need to extend existing game theoretic schemes to handle the incomplete information situations, possibly by using Bayesian game approaches. Thirdly, incentive mechanisms for wireless P2P systems are largely unexplored. An obvious approach is to extend wired Internet-based incentive schemes to a wireless environment. However, a wireless network exhibits many unique features such as time-varying channel quality, energy efficiency concerns, and mobility that are not found in a wired Internet environment. Thus, it is a challenge to design new incentive schemes that are suitable for a wireless network. Finally, reputation based incentive schemes are currently designed without much regard to trust management. It is desirable to integrate efficient trust management schemes with a reputation system to form a comprehensive unit that allows peers to perform judicious peer selection as well as identify malicious colluders.

On the security front, there are even more important problems to be tackled. Indeed, all of the security attacks that we described still need much research. Churn attacks are still relatively easy to launch but still can disrupt the normal operations of a P2P system. A promising avenue of research along this line is to devise effective technique to raise the barrier of joining the system on the part of the peer but not to the extent that some honest peers are deterred. Similarly, again due to the open nature of a P2P network, poisoning seems to be inevitable. The issue really is how soon such a poison-injecting peer can be identified and expelled from the system. Sybil attacks have received intensive research attention recently and many novel solutions based on various cryptographic techniques have been proposed. Nevertheless, as the hardware and software capabilities of a malicious peer become more advance, existing techniques may not be effective. Worms and malwares distribution continue to be major headaches in the Internet as a whole and P2P systems are no exception. However, an interesting research problem is to leverage the cooperativeness of participating peers to collaborative combat worms and malwares spreading. Cheating in a P2P MMOG is of interest to many commercial attackers. Thus, it is of great financial impact to successfully track down cheating peers. Perhaps a useful approach is to provide game-related incentive (or even in real monetary terms) for honest peers to proactively report anomalies exhibited by topologically close malicious peers.

6 Summary

P2P systems are autonomic in nature because participating peers can join and leave at will. To ensure sustained operations in a P2P network, cooperation among peers and a secure environment are inevitable. In this chapter, we first provide a brief survey of state-of-the-art techniques to provide incentives for cooperation. In general, systems that involve payment would be more difficult to implement because it is not

trivial to design a global “currency” for use in such systems. Furthermore, security requirement would be high because the payment could be forged by malicious peers. On the other hand, exchange based or reciprocity based are easier to implement and hence, are more scalable. The major crux is that there is much less *state information* to be kept by each peer. More importantly, the accuracy of such state information (e.g., reputation) does not need to be absolutely very high. Thus, we expect that future P2P file sharing would still be based on similar approaches.

For the security issue, a P2P system faces many unique challenges. Churn attacks are a P2P version of denial-of-service attacks in that malicious peers deliberately join and leave the system frequently, inducing a large amount of processing and communication overheads for population maintenance. Poisoning and Pollution occur when some malicious peers intentionally injects falsified data (e.g., unusable files) into the shared pool of contents. The most commonly seen attacks are the various forms of sybil attacks. Because many P2P systems rely on voting to make collective decisions, it is important that peer IDs cannot be easily forged; otherwise, a peer can control a large portion of peer IDs and out-vote other honest peers. In many P2P file sharing systems, peers download files without much checking and thus, such a platform is an idea environment for spreading worms. Related to poisoning and worms, honest peers could inadvertently downloaded files encapsulating malwares such as virus codes. Many massively MMOGs are moving toward a P2P architecture from a traditional centralized client-server structure. However, such a move also gives away the centralized control over peers’ actions and the associated effective security measures. In effect, as the game state updating and communication tasks are handled by the peers themselves, some malicious peers can cheat by launching many of the above attacks. As in a traditional Internet computing environment, security is an ongoing battle and all of the problems we described in this chapter still need much research.

References

1. BitComet, <http://www.bitcomet.com/>, 2008.
2. BitTorrent, <http://www.bittorrent.com/>, 2008.
3. eDonkey2000, <http://www.edonkey2000.com/>, 2005 (defunct).
4. eMule, <http://www.emule-project.net/>, 2008.
5. FreeNet, <http://freenetproject.org/>, 2008.
6. giFT Project, <http://gift.sourceforge.net/>, 2008.
7. Gnutella, <http://gnutella.wego.com/>, 2008.
8. GreenTea, <http://www.greenteatech.com/>, 2008.
9. HKCSL, <http://one2free.hkcs1.com/eng/main/index.jsp>, 2005.
10. iMesh, <http://www.imesh.com/>, 2008.
11. Joost, <http://www.joost.com/>, 2008.
12. KaZaA, <http://www.kazaa.com/>, 2008.
13. Limewire, <http://www.limewire.org/>, 2008.
14. Napster, <http://www.napster.com/>, 2002 (bankruptcy).
15. PPLive, <http://www.pplive.com/>, 2008.
16. Skype, <http://www.skype.com/>, 2008.

17. Slyck.com, <http://www.slyck.com/news.php?story=574>, 2005.
18. Tribler, <http://www.tribler.org/>, 2008.
19. Washington Times Online, <http://www.washtimes.com/technology/20040303-094741-3574r.htm>, 2005.
20. World of Warcraft, <http://www.worldofwarcraft.com/>, 2008.
21. WinMX, <http://winmx.com/>, 2005
22. K. G. Anagnostakis and M. B. Greenwald, "Exchange-Based Incentive Mechanisms for Peer-to-Peer File Sharing," *Proc. 24th Int'l Conference on Distributed Computing Systems*, 2004.
23. S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, no. 4, Dec. 2004, pp. 335–371.
24. J. U. Becker and M. Clement, "The Economic Rationale of Offering Media Files in Peer-to-Peer Networks," *Proc. 37th Hawaii Int'l Conference on System Sciences*, 2004.
25. S. Capkun, J.-P. Hubaux, and L. Buttyan, "Mobility Helps Peer-to-Peer Security," *IEEE Transactions on Mobile Computing*, vol. 5, no. 1, Jan. 2006, pp. 43–51.
26. D. Cerri, A. Ghioni, S. Paraboschi, and S. Tiraboschi, "ID Mapping Attacks in P2P Networks," *Proc. IEEE GLOBECOM 2005*, pp. 1785–1790.
27. G. Chen and R. S. Gray, "Simulating Non-Scanning Worms on Peer-to-Peer Networks," *Proc. 1st ACM Int'l Conf. Scalable Information Systems*, May 2006.
28. N. Christin, A. S. Weigend, and J. Chuang, "Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks," *Proc. ACM EC 2005*, June 2005, pp. 68–77.
29. I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Proc. Workshop on Design Issues in Anonymous and Unobservability*, July 2000.
30. B. Cohen, "Incentives Build Robustness in BitTorrent," *Proc. Workshop on Economics of Peer-to-Peer Systems*, June 2003.
31. A. B. Corman, S. Douglas, P. Schachte, and V. Teague, "A Secure Event Agreement (SEA) Protocol for Peer-to-Peer Games," *Proc. 1st IEEE Int'l Conf. Availability, Reliability and Security (ARES 2006)*.
32. G. Danezis and R. Anderson, "The Economics of Resisting Censorship," *IEEE Security and Privacy*, Jan./Feb. 2005, pp. 45–50.
33. S. Daswani and A. Fisk, *GUESS Protocol Specification*, http://groups.yahoo.com/group/the_gdf/files/Proposals/GUESS/guess.01.txt.
34. N. Daswani and H. Garcia-Molina, "Pong-Cache Poisoning in GUESS," *Proc. ACM CCS 2004*, Oct. 2004, pp. 98–109.
35. M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-Riding and Whitewashing in Peer-to-Peer Systems," *Proc. 2004 SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems*, Aug. 2004, pp. 228–235.
36. M. Feldman and J. Chuang, "Overcoming Free-Riding Behavior in Peer-to-Peer Systems," *ACM SIGCOMM Exchanges*, vol. 5, no. 4, July 2005, pp. 41–50.
37. M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust Incentive Techniques for Peer-to-Peer Networks," *Proc. 5th ACM conference on Electronic Commerce*, pp. 102–111, May 2004.
38. A. Friedman, "Good Neighbors Can Make Good Fences: A Peer-to-Peer User Security System," *IEEE Technology and Society Magazine*, Spring 2007, pp. 17–24.
39. C. GauthierDickey, D. Zappala, V. Lo, and J. Marr, "Low-Latency Cheat-Proof Event Ordering for Peer-to-Peer Games," *Proc. Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2004.
40. I. Gupta, K. Birman, P. Linga, A. Demers, and R. Van Renesse, "Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead," *Proc. 2nd Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 81–86.
41. R. Gupta and A. K. Somani, "A Pricing Strategy For Incentivizing Selfish Nodes To Share Resources In Peer-to-Peer (P2P) Networks," *Proc. IEEE International Conference on Networks*, Nov. 2004.

42. D. Hausheer, N. C. Liebau, A. Mauthe, R. Steinmetz, and B. Stiller, "Token Based Accounting and Distributed Pricing to Introduce Market Mechanisms in a Peer-to-Peer File Sharing Scenario," *Proc. Third International Conference on Peer-to-Peer Computing*, 2003.
43. D. Hausheer and B. Stiller, "Decentralized Auction-based Pricing with PeerMart," *Proc. 9th IFIP/IEEE International Symposium on Integrated Network Management*, May 2005.
44. Z. Jia, S. Tiange, H. Liansheng, and D. Yiqi, "A New Micropayment Protocol Based on P2P Networks," *Proc. 2005 IEEE Int'l Conference on e-Business Engineering*, 2005.
45. P. Kabus and A. P. Buchmann, "Design of a Cheat-Resistant P2P Online Gaming System," *Proc. ACM DIMEA 2007*, pp. 113–120.
46. A. Kalafut, A. Acharya, and M. Gupta, "A Study of Malware in Peer-to-Peer Networks," *Proc. ACM IMC 2006*, Oct. 2006, pp. 327–332.
47. M. A. Konrath, M. P. Barcellos, and R. B. Mansilha, "Attacking a Swarm with a Band of Liars: Evaluating the Impact of Attacks on BitTorrent," *Proc. 7th IEEE Int'l Conf. Peer-to-Peer Computing*, 2007, pp. 37–44.
48. R. Krishnana, M. D. Smith, and R. Telang, "The Economics of Peer-to-Peer Networks," *Journal of Information Technology Theory and Application*, vol. 5, no. 3, pp. 31–44, 2003.
49. R. Kumar, D. D. Yao, A. Bagchi, K. W. Ross, and D. Rubenstein, "Fluid Modeling of Pollution Proliferation in P2P Networks," *Proc. ACM SIGMetrics 2006*, June 2006, pp. 335–346.
50. G. Lawton, "Is Peer-to-Peer Secure Enough for Corporate Use?" *IEEE Computer*, Jan. 2004, pp. 22–25.
51. P. Linga, I. Gupta, and K. Birman, "A Churn-Resistant Peer-to-Peer Web Caching System," *Proc. ACM SSRS 2003*, Oct. 2003.
52. R. T. B. Ma, S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau, "A Game Theoretic Approach to Provide Incentive and Service Differentiation in P2P Networks," *Proc. SIGMETRICS*, June 2004, pp. 189–198.
53. R. T. B. Ma, S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau, "An Incentive Mechanism for P2P Networks," *Proc. 24th Int'l Conference on Distributed Computing Systems*, 2004.
54. P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proc. 1st Int'l Workshop on Peer-to-Peer Systems (IPTPS 2002)*, pp. 53–65.
55. M. J. Osborne, *An Introduction to Game Theory*, Oxford University Press, 2004.
56. V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," *Proc. 4th Int'l Workshop on Peer-to-Peer Systems*, pp. 127–140, Feb. 2005.
57. D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," *Proc. SIGCOMM*, pp. 367–377, Sept. 2004.
58. K. Ranganathan, M. Ripeanu, A. Sarin and I. Foster, "To Share or not to Share: An Analysis of Incentives to Contribute in Collaborative File-Sharing Environments," *Proc. Workshop on Economics of Peer-to-Peer systems*, June 2003.
59. M. Roussopoulos, M. Baker, D. Rosenthal, T. J. Giuli, P. Maniatis, and J. Mogul, "2 P2P or Not 2 P2P?," *Proceedings of the Third International Workshop on Peer-to-Peer Systems (IPTPS '04)*, Feb. 2004. La Jolla, CA.
60. A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object address and routing for large-scale peer-to-peer systems," *Proc. IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, Nov. 2001.
61. S. Sanghavi and B. Hajek, "A New Mechanism for the Free-Rider Problem," *Proc. SIGCOMM 2005 Workshop*, Aug. 2005.
62. T. C. Schelling, *Micromotives and Macrobehavior*, W. W. Norton & Company, 1978.
63. S. Shin, J. Jung, and H. Balakrishnan, "Malware Prevalence in the KaZaA File-Sharing Network," *Proc. ACM IMC 2006*, Oct. 2006, pp. 333–338.
64. K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," *Proc. SIGCOMM*, vol. 34, no. 4, pp. 107–120, Aug. 2004.

65. I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM 2001*, Aug. 2001, pp. 149–160.
66. Q. Sun and H. Garcia-Molina, "SLIC: A Selfish Link-based Incentive Mechanism for Unstructured Peer-to-Peer Networks," *Proc. 24th Int'l Conference on Distributed Computing Systems*, 2004.
67. J. Van Der Werwe, D. Dawoud, and S. McDonald, "A Survey on Peer-to-Peer Key Management for Mobile Ad Hoc Networks," *ACM Computing Surveys*, vol. 39, no. 1, Apr. 2007.
68. V. Venkataraman, P. Francis, and J. Calandrino, "Chunkspread: Multi-Tree Unstructured Peer-to-Peer Multicast," *Proc. 5th Int'l Workshop on Peer-to-Peer Systems*, Feb. 2006.
69. W. Wang and B. Li, "Market-driven Bandwidth Allocation in Selfish Overlay Networks," *Proc. IEEE INFOCOM 2005*, Mar. 2005.
70. F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast," *Proc. ICDCS*, June 2007.
71. S. D. Webb and S. Soh, "Cheating in Networked Computer Games—A Review," *Proc. ACM DIMEA 2007*, pp. 105–112.
72. J. D. Weisz, S. Kiesler, H. Zhang, Y. Ren, R. E. Kraut, and J. A. Konstan, "Watching Together: Integrating Text Chat with Video," *Proc. SIGCHI*, pp. 877–886, Apr.–May 2007.
73. K. Wongrujira and A. Seneviratne, "Monetary Incentive with Reputation for Virtual Market-Place Based P2P," *Proc. CoNEXT'05*, Oct. 2005.
74. B. Yang and H. Garcia-Molina, "PPay: Micropayments for Peer-to-Peer Systems," *Proc. 10th ACM Conference on Computer and Communication Security*, pp. 300–310, 2003.
75. W. Yu, S. Chellappan, X. Wang, and D. Xuan, "On Defending Peer-to-Peer System-Based Active Worm Attacks," *Proc. IEEE GLOBECOM 2005*, pp. 1757–1761.
76. H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: Defending Against Sybil Attacks via Social Networks," *Proc. ACM SIGCOMM 2006*, Sept. 2006, pp. 267–278.
77. X. Zhang, J. Liu, B. Lim, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," *Proc. INFOCOM*, vol. 3, pp. 2102–2111, Mar. 2005.
78. M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "Optimizing the Throughput of Data-Driven based Streaming in Heterogeneous Overlay Network," *Proc. 13th Int'l Multimedia Modeling Conf.*, Jan. 2007.
79. M. Zhang, L. Zhao, Y. Tang, J.-G. Luo, and S.-Q. Yang, "Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet," *Proc. Workshop on Advances in Peer-to-Peer Multimedia Streaming*, pp. 21–28, Nov. 2005.

Part II

Autonomic Networking

Toward Autonomic Networks: Knowledge Management and Self-Stabilization

Raouf Boutaba, Jin Xiao, and Qi Zhang

Abstract Autonomic networks present a fundamental shift in the design philosophy of networks and systems. Thus far, research has focused on addressing the self-management properties. In this book chapter, we illuminate two important issues in autonomic networks that are seldom addressed: knowledge management and self-stabilization. Through in-depth discussion on their concepts, challenges, and relevant works, we show that exploration in these issues is not only necessary but also critical to the success of autonomic networks.

1 Autonomic Networking

1.1 Introduction

Information and networking technology has become an intrinsic part of our daily lives and business activities. We are accustomed to diverse means of digital information access and communication regardless of geographical locations and physical distances. The information and networking technology is so vital to human society today, any malfunctions, whether in performance or due to fault, directly impacts our personal lives, enterprise operations, and global economy. This way of life is supported through vast deployments of communication and network infrastructures and its growth is sustained by the ever increase in size and complexity of these infrastructures. Since the inception of networks and communication systems, network management has been pivotal in ensuring their correct and efficient operation in aspects of configuration, fault, accounting, performance, and security. Traditionally, network management is conducted in centralized fashion with heavy reliance on manual administration. However, with the rising complexity and scale of these infrastructures, this approach becomes wholly ineffective in practice. The networks and software systems have grown beyond the limit of manual administration due to their complexity in design, pervasiveness in distribution and the dynamicity of

R. Boutaba (✉)

David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
e-mail: rboutaba@cs.uwaterloo.ca

runtime environment. Today, the rising cost of networking management and operations far out strides that of network expansion [41]. And at the same time, there is rising pressure in making network management cheaper and more effective.

Over the years, much research efforts have been spent on distributing management intelligence into the networks and communication systems (e.g. management by delegation, policy-based management, mobile agents), with the eventual goal of making network management autonomous and generally free of human reliance. These efforts have recently culminated in the concepts of autonomic computing [18] and networking [31]. This marks a fundamental shift in the design philosophy of networks and systems. No longer are the networks primitive and the management systems omnipotent and all encompassing, a large degree of self-awareness and self-governance must be realized.

Autonomic networking aims to engender networks that exhibit the following self-management properties:

- Self-configuration: the entities can automate system configuration following high-level specifications, and can self-organize into desirable structures and/or patterns.
- Self-optimization: the entities constantly seek improvement to their performance and efficiency, and able to adapt to changing environment without direct human input.
- Self-healing: the entities can automatically detect, diagnose, and recover from faults as the result of internal errors or external inconsistencies.
- Self-protection: the entities can automatically defend against malicious attacks or isolate the attacks to prevent systemwide failures.

Figure 1 shows the anatomy of an autonomic component whereby the autonomic manager interacts with the managed elements and its surroundings by taking inputs from the external environment, applying analysis and reasoning logic, generating

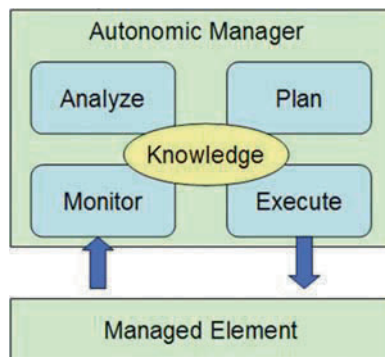


Fig. 1 The anatomy of autonomic manager

corresponding actions, and executing these actions as output. This workflow fits well with the classic monitoring and control loop of network management, where monitored data from the networks are used to determine the appropriate management decisions and then translated into corresponding control actions.

To date, there has been a flurry of research activities on addressing the self-management properties in various networking contexts, often relating to one of the five management functional areas. However, not much attention has been paid to the fundamental issues of knowledge management and self-stabilization in autonomic networking. In fact, they are the missing facets that enables "autonomicity" in autonomic computing and networking. More precisely, knowledge management is the fundamental underpin of self-* behaviors (i.e. the baseline inputs), while self-stabilization is very much an essential management and control objective that should encompass the self-managing behaviors of the networks and systems. Hence, the goal of this book chapter is to explore these two topics in considerable depth, and discuss the key concepts, challenges, and related works.

1.2 Knowledge and Stability

Knowledge is central to the operation of autonomic networking, as depicted in Fig. 1. The knowledge an autonomic system is required to gather and formulate far exceeds the level of data representation and interpretation capability of today's network management applications and distributed systems. There is a need for knowledge representation of the environment, various processes, services and objects, the autonomic component, the states the system could take on, the interactions a component could have, the properties the system and components are associated with, and the features/services the system or component provides. There is also a need for the representation of high-level objectives, system behaviors, and rules. The problem is further complicated by the wide range of contexts such knowledge could be taken from, the need for common information representation, and the possibility of varied semantics and interpretations. Nevertheless, an efficient and comprehensive knowledge management system is the critical foundation for self-management, self-awareness, and self-stabilization.

In this context, knowledge management refers to *the activities of collecting, storing, delivering, and reasoning about the knowledge of the networks, in order to benefit network operations and management*. Knowledge management is envisioned to have a broad impact on the future of autonomic networks. In particular, Clark et al. in his position paper [8] describes a global knowledge management infrastructure called knowledge plane, capable of automating network management tasks and driving autonomic behavior of individual network entities, at the local and global scale. Although various aspects of knowledge management have received much attention (e.g., monitoring), there lacks formal structure to knowledge management research as a whole.

Today's networking infrastructure is carefully planned and engineered to ensure its runtime efficiency and stability. The autonomic networking philosophy presents an environment of large distributed control, often lacking the effectiveness and thus the feasibility of centralized planning. Hence the issue of distributed self-stabilization is critical in this context. Can networks operating on self-motivated behaviors and limited information be stable? Under what conditions could stabilization be obtainable? How can we design networks that can recover from unstable configurations? these are but some of the important questions that sorely lack solution at present. The concept of self-stabilization, as first proposed by Dijkstra [10] for distributed systems still bears much relevance in the autonomic networking context. Although there are much research on self-stabilization in the past [12], they are focused on algorithm designs and system/protocol engineering. In this chapter, we examine self-stabilization from the unique perspective of autonomic networks, and as we will discuss, the accuracy and timeliness by which knowledge could be obtained about the environment is critical to the feasibility of self-stabilizing behaviors of the components and the global system.

The rest of the chapter is organized as follows. Section 2 covers the topic of knowledge management, presenting its various aspects and related works. Section 3 addresses the stability issue, relating the concept to research on game theory and discuss the properties of self-stabilization in autonomic networking. Section 8 concludes the book chapter.

2 Knowledge Management

In this section, we first present an overview of knowledge management concept, in comparison with knowledge management in organizational management research. Then, we discuss individual aspects of knowledge management and the related works.

2.1 Knowledge Management Concepts

Knowledge management is well-studied in organizational management. In this context, a distinction is often made among the following terms: *data* as raw numbers and facts obtained from observations; *information* as the outcome of analyzing relationships between data; and *knowledge* as organized information in an individual's mind that can be used for reasoning and explanation [1]. The difference between these three terms could be subtle. For instance, it is arguable that even elementary data is a result of data identification and collection process. Furthermore, there could be some intangible mapping between information and knowledge when it is stored in an individual's mind (information-to-knowledge) and presented by the individual in text, graphics or other semantic forms (knowledge-to-information and

knowledge-to-data). Nevertheless, it is necessary for a knowledge management system to consider a combination of knowledge, information, and data [4].

Knowledge management in organizational settings typically refers to a process [1] involving various activities: creating, storing/retrieving, transferring, and applying knowledge to applications. In this regard, a knowledge management system is an information system responsible for managing organizational knowledge. Thus, knowledge management is sometimes equated to information management. We argue based on the definitions of data, information and knowledge that knowledge management encompasses all the activities in information management, plus the activity of cognitive processing. Cognitive processing is the ability to correlate, reason, and infer about knowledge.

In the context of networks, knowledge often refers to models and information regarding the network capabilities, environmental constraints, business goals, and policies [19]. It is possible to classify network knowledge according to the following three criteria:

- **Meta-knowledge vs. instance-knowledge:** analogous to meta-data (i.e., schema) and data items in databases, there is a difference between meta-knowledge and instance-knowledge in knowledge management. Meta-knowledge describes the structure and relationship between knowledge objects, while instance-knowledge is the realization of meta-knowledge. Ontology is often used in meta-knowledge representation [15]. For example, various semantics and domain-specific concepts can be represented through ontologies [20, 28].
- **Functional areas and management levels:** management information is traditionally organized into layers: business, service, network, and device. This presents a natural grouping for knowledge management. Furthermore, the five classic network management functional areas—fault, configuration, accounting, performance, and security—provide a functional grouping of knowledge. These two groupings of knowledge are complimentary, as shown in current network management designs.
- **Temporal and spatial knowledge:** temporal wise, knowledge can be scoped by its associated time length. Spatial wise, knowledge can be scoped by its geographical dimensions. For example, IP address and routing table values are scoped to individual routers and hosts, while Border Gateway Protocol (BGP) policies are scoped with entire network domains. Such scoping is required for particular management activities (e.g., fault diagnosis and security analysis).

Context also plays an important role in knowledge management. Context refers to a collection of information that describes the operating environment of the system. Context can be raw data or information interpreted at multiple levels. It is generally agreed that achieving context-awareness is essential to achieving autonomic network management [11, 21]. From a conceptual point of view, context management is identical to knowledge management, since managing context also involves the tasks of collecting, representing, and interpreting contextual information, and applying the contextual knowledge to drive adaptive and autonomic behavior of network entities.

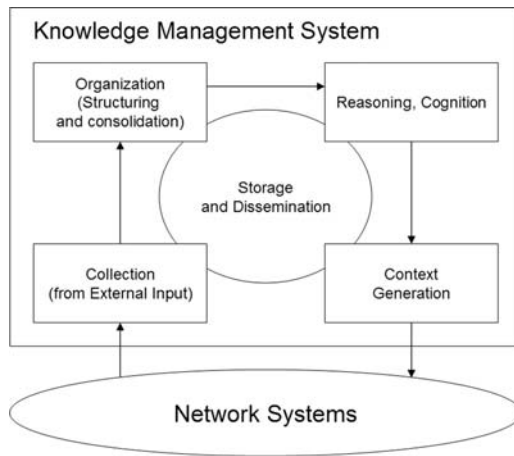


Fig. 2 Knowledge management for autonomic networks

Figure 2 depicts the set of activities in knowledge management. The reader should bear in mind that a knowledge management system is an integral part of the autonomic networking infrastructure. Collection, organization, cognition, storage and dissemination, and context generation are the main aspects of knowledge management. In *collection*, input data is acquired from the networks. The data come from sources at multiple levels, such as measurement data from network sensors, business level policies, and objectives from network operators. The goal of *organization* is to structure and integrate the input data from diverse sources and store them as information, preparing them for reasoning and cognition. The *cognition* process correlates, reasons, and infers information to generate knowledge about the system. The resulting knowledge could then be represented to the outside as information in a specific *context*. The *Storage and Dissemination* process underpins the other activities. It is tasked with data, information and knowledge storage and retrieval in distributed and efficient ways. In the autonomic networking infrastructure, knowledge management is a continuous process in which network data are processed by the knowledge management system in order to guide the autonomic behaviors of the network.

2.2 Knowledge Management in Autonomic Networks

In this section, we expound the knowledge management activities. We first discuss knowledge modeling and representation, then describe each of the aforementioned knowledge management aspects.

2.2.1 Knowledge Modeling and Representation

Traditionally, knowledge representation has been studied in the area of artificial intelligence. The goal is to represent knowledge in a machine-understandable way

in which knowledge can be processed and reasoned about. Conceptually, knowledge can be represented as a set of concepts; each concept may be related to other concepts through particular relationships. Ontology is then a formal and structured representation of knowledge and specifies how concepts are related. Ontology is specified using ontology languages. The most predominate ontology language is the Web Ontology Language (OWL) [40]. Standardized by W3C, OWL is a XML-based ontology language supporting knowledge correlation and reasoning. More specifically, OWL categorizes individual concepts using classes. Each class can have associated properties such as domain and constraints, and can be associated and correlated to other concepts based on these properties. Currently, the OWL language provides three subset languages corresponding to three levels of semantic expressiveness: OWL Lite is for specifying simple constraints, OWL DL allows for description logic and guarantees completeness and decidability, and OWL full is the most expressive but may not be computational feasible.

In fact, the concept of objects, properties and relations are not foreign to network management. Many existing information models such as Structured Management Information (SMI) and Guidelines for Definition of Management Objects (GDMO) already have these notions. However, SMI and GDMOs are not well-suited for knowledge representation due to two reasons. First, they lack semantic capability. Autonomic networks operate across heterogeneous environments, each with its own information representation, structure, and terminology. To achieve interoperability, the knowledge representation must operate on the semantic level rather than syntactical level. Secondly, they lack reasoning capability. Due to lack of formal semantic models and reasoning constructs, one could not reason or infer about the information provided by SMI and GDMO. To address these problem, several research effort have been devoted to represent network management knowledge using OWL, since OWL is formally defined and enables reasoning, and it also eases the integration between different information models.

In particular, Lopez de Vergara et al. [24] compare the expressiveness of different management languages and propose new methodologies for mapping ontological definitions. They also describe equivalent construct of common objects, attribute properties and constraints in three management information models (SMIv2 GDMO, and CIM) in OWL [25]. Several ontology mapping techniques such as lexical-b, taxonomy-b, and logical-based matching techniques have been studied. Kenny et al. [21] computes similarity scores for matching ontological concepts using model-based matching on semantic aspects. All of the above techniques are either performed manually or require software assistance.

Using ontology to capture semantic information has also been applied to other information models. In particular, Strassner et al. propose to augment Directory Enabled Networking-next generation (DEN-ng) information model with ontology extensions [42]. Specifically, DEN-ng information model is based on the Unified Modeling Language. It defines a set of models capturing various aspects of the managed entities, called views. The ontology extension of DEN-ng supports interoperability among information models, as well as reasoning using first-order logic or other machine learning techniques.

Overall, representing knowledge using ontology is a promising direction of research. Ontology offers the salient advantages of formal methodologies, decidability, and interoperability.

2.2.2 knowledge Collection

Knowledge management systems acquire data from diverse sources: network information bases, online monitors, system and application logs, and other knowledge management systems. Excluding human-specified data, most of the knowledge collection tasks are typically carried out by network monitoring systems such as network sensors and agents. Some existing network management protocols/tools such as SNMP and NetFlow, already provide built-in monitoring functionality. However, traditional network monitoring is not adequate for knowledge management systems, for several reasons [27]. First, the information that can be monitored is usually hard-coded according to a standard information model. This is problematic because in the context of knowledge management, the knowledge that needs to be captured is diverse and constantly changing according to its operating environment. Second, most of the existing network monitoring systems are centralized, suffering from scalability and single-point-of-failure issues. Third, existing monitoring systems often conduct monitoring at some fixed rate, regardless of the actual network conditions. Hence when network condition is stable, the monitoring system generates excessive data; when a significant event such a failure occurs, the system often fails to capture the important characteristics of the event due to coarse monitoring granularity. Thus, a suitable monitoring infrastructure for network knowledge management must be distributed, coordinated, adaptive, and flexible.

Recent work on monitoring address some of the issues aforementioned. In particular, there has been a significant interest in building distributed monitoring systems for large-scale networks [26, 44]. For instance, iPlane [26] is a distributed monitoring service that provide end-to-end path performance estimation to end hosts over the Internet. To reduce the measurement overhead, iPlane selectively probes a path to a representative node with each prefix (BGP atoms). The iPlane measurement is coordinated, in that each probe is responsible for a set of prefixes according to a clustering algorithm. The primary measurement tool used by iPlane is TraceRoute. Using a set of heuristics, iPlane is able to determine router level link attributes such as loss rate, capacity, and bandwidth availability. This information can then be used to predict end-to-end path performance. Although it operates primarily at interdomain level, iPlane is an example of how distributed and coordinated monitoring could be achieved. Sophia [44] is a distributed monitoring application that provides a declarative query language based on Prolog. Furthermore, the notion of time and location are first class elements in query expressions, which guarantee precise interpretation of queries. Sophia handles churns and failures by allowing logic “holes” in query evaluation, thus allowing better performance at the cost of completeness. it also employs other means of performance optimization such as caching, query planning, and scheduling.

2.2.3 Knowledge Storage and Dissemination

One of the key issues in designing a network knowledge management system is determine where knowledge should be stored and how knowledge can be delivered. Traditional network management systems often store information in centralized information stores, which raises serious scalability and performance issues in large distributed setting. Recent works have begun to investigate distributed network management. This issue is even more important to autonomic networks, where autonomic elements heavily rely on knowledge management infrastructure to acquire their operating context, and achieve self-management. Therefore, scalable knowledge storage and efficient knowledge delivery is central to a knowledge management system. In this section, we briefly summarize several recent research efforts.

Mulvenna et al. [32] introduces the concept of knowledge networks, which is an overlay network responsible for distributing knowledge across the network. They draw analogy to the human nervous system which coordinates and organizes the delivery of stimuli in a timely manner. The proposal is based on the observation that local knowledge may not be sufficient for driving autonomic behavior of components; thus, a global infrastructure is required. Such an infrastructure must be scalable, robust to failure, and self-organizing when network structures change. They envisioned knowledge networks as an intermediate layer between the applications and the networks to both enable access to contextual information and allow better coordination between components. Furthermore, to achieve efficiency and load balancing, they argue that knowledge should be distributed according to the source and demand of the knowledge. The “load field” is a virtual landscape that summarizes the distribution of knowledge across the network. The knowledge network can hence re-distribute load based on the global load situation. To date, there is little implementation to their conceptual knowledge framework. The authors suggest that many practical issues remain.

Lewis et al. [21, 23] present a network Knowledge Delivery Service which provides efficient delivery of network knowledge to components for the purpose of self-adaptation. Implemented as a Knowledge Delivery Network (KDN), KDN is based on Content-Based Networks (CBNs), which offer a publish–subscribe interface for information delivery. The knowledge in KDN is stored and represented using OWL, and subscription queries can be specified using predicate logic. To achieve scalability, robustness, and self-organization in KDN, the knowledge nodes in a KDN are organized as a peer-to-peer (P2P) overlay network. Furthermore, knowledge nodes are dynamically clustered, based on knowledge provider, knowledge subscription, knowledge semantics, and administrative restrictions. Dynamic clustering of knowledge nodes allows a KDN to achieve substantial improvement in performance and efficiency. For evaluation, the authors build a prototype CBN and compared the performance of several ontology reasoners. The experimental results show that initial loading and reasoning of ontology is both memory and computationally expensive, but subsequent querying operations can be done very efficiently. The authors postulate that ontological relationships and axioms could be used to determine semantic clustering of knowledge nodes. However, details of the clustering algorithms were not described.

Astrolabe [43] is a hierarchically organized P2P query processing system. Nodes in Astrolabe are hierarchically organized into zones. Each node maintains its individual management information bases (MIBs), as well as the MIBs of the zones to which it belongs. A gossiping protocol is used to dynamically synchronize MIBs. Astrolabe allows specification of aggregation functions for each query. These aggregation functions determine how information at each hierarchical level is aggregated. This powerful feature of Astrolabe not only allows it to support more expressive queries, but also enables a variety of communication paradigms such as caching, multicasting, and publish–subscribe. The main drawback of Astrolabe is that topology must be manually maintained by administrators. Inspired by Astrolabe, Yalagandula et al. implement a Scalable Distributed Information Management System (SDIMS) [46]. Similar to Astrolabe, information in SDIMS is organized in a tree and aggregated using aggregation functions. However, instead of maintaining topology manually, aggregation trees are embedded in a locality-aware DHT structure. Furthermore, administrative isolation can be achieved through flexible embedding of trees into DHT. SDIMS also achieves robustness by providing replication along the aggregation tree and enabling lazy and on-demand re-aggregation.

2.2.4 Reasoning and Cognition

In biological systems, the cognition process consists of mental activities such as awareness, perception, reasoning, and judgment [35]. In the context of networks, cognition refers to the ability of the system to interpret its objectives, reasoning about its current state, and planning for future actions based on its current knowledge. These issues have been studied extensively in the field of artificial intelligence. In particular, knowledge reasoning and machine learning are useful concepts in knowledge management.

Knowledge reasoning refers to the derivation of conclusions through logic. It is a direct feature offered by logical representation of knowledge. Two types of logical reasonings are typically discussed: (1) Deductive reasoning, in which derived conclusions must be true when the reasons supporting the conclusion are true; and (2) Inductive reasoning, in which conclusions is true with certain probability when the supporting reason is true. One particular issue with reasoning is the trade-off between expressiveness and decidability of knowledge description languages: Expressive languages usually lead to undecidable reasoning problems. For instance, first-order logic (FOL) used in OWL-full is known for its expressiveness for representing real-world ontology, but reasoning problems for FOL is usually undecidable. On the contrary, description logic (DL) used in OWL-DL is more restrictive than FOL, but reasoning problems in DL is tractable.

Research on machine learning aims at designing algorithms to induce rules and patterns from known information. Such rules and patterns can be used to understand current system behaviors or predict future state of the system. Machine learning techniques can be generally divided into four categories: (1) Supervised learning, in

which the system learns a function that maps input data to desired outputs. Supervised learning typically requires a training process, in which the system observes both inputs and outputs from a training set and deduces the relationship between them; (2) Unsupervised learning, in which only a set of inputs is provided to the system without corresponding outputs. Clustering problems are a type of unsupervised learning problems, since the algorithm must determine the solutions based solely on the inputs; (3) Semi-supervised learning, which is similar to supervised learning except not every output is known for every input; and (4) Reinforcement learning—in which the algorithm learns to take proper actions based on observations, in order to maximize a long-term reward function. Besides these general categories, there is also a distinction between off-line and online learning algorithms. Generally speaking, online learning algorithms are more suitable for the purpose of knowledge learning in networks, as network operating environments are highly dynamic.

2.2.5 Context Generation

It is important to inform network components about their context (i.e., context-awareness), such that self-management activities could be conducted. The key issues here are to determine what communication interface and paradigm should be provided, what information should be delivered. Clearly, different components require different information, according to their operating context and environment. Secondly, every component demands information at different rate, and some may require on-demand notifications, when certain events occur. Although many current standard network management architectures offers some of these features, there still lacks flexibility in the information interface and the delivery rate to meet the requirement of each individual components.

Many research in this area adopt a distributed database-like approach. The aim is to provide a flexible query and update interface such that each individual component could obtain its information in context. For example, PIER [17] is a P2P database system of DHTs. In PIER, each node maintains a local query processor and a local storage of database records which consists of key value pairs. Each object in PIER is uniquely identified by the combination of namespace, resourceID and instanceID. The namespace and resourceID can be used to compute the DHT key using a hash function. Queries are answered by executing query plans as in traditional database systems. Clearly, database-like query-update interface is not the only possible strategy. Publish-subscribe and trigger-based interfaces are also used for disseminating knowledge. Hellerstein et al. [16] studied the design of aggregated triggers to reduce packet exchange overhead.

Proving context to network components also implies guiding network monitoring devices on monitoring behavior. As mentioned in Sect. 3.1, self-adaptive monitoring is essential to achieve scalability and efficiency in collecting large volume of network information. Given the current state of the network, the monitoring devices can determine how data collection rate should be adjusted.

2.2.6 Knowledge Management as the Foundation for Autonomy

Knowledge management is an essential component of autonomic networks. It is strongly related to self-awareness. Through the knowledge collection, organization, reasoning, and context generation activities, the components are made aware of their surrounding environment and their context, which is key to drive their self-management behaviors. For example, fault diagnosis and troubleshooting are concerned with identifying root causes for faults. This process involves detecting, correlating, and learning about fault symptoms, and reasoning about potential problems. All of these processes are inherently supported by the knowledge management system.

As we will see in Sect. 3, knowledge management is also an essential enabler of the self-stabilization behavior of the global system.

3 Self-Stabilization in Autonomic Networks

3.1 *Self-Stabilization and Autonomic Networks*

In his seminal work [10], Dijkstra describes a self-stabilizing system as a distributed system where individual component behaviors are determined by a subset of global system state that is known to the component. The global system must exhibit the property of “regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps.” Two key properties are outlined: (1) the system can initialize in any state and (2) the system can always recover from transient faults. The concept of “legitimate” states is important to self-stabilization. A legitimate state denotes an operational state of a system that is within the design consideration of the system. Thus, as long as the system is in a legitimate state, its operation is expected or bounded. The self-stabilization concept recognizes that the set of all possible states of a system is rather large in the face of arbitrary faults and attacks, most of which are outside the specification of the system design. When discussing self-stabilization, the concept of closure and convergence [38] are often mentioned. The closure property describes that if the system is stable (i.e., in a legitimate state), after a number of executions it will remain stable unless perturbed by external force. The convergence property states that if the system is unstable (i.e. not in a legitimate state), it has a tendency to be stable over time. The length of time that must elapse for the system to move from an illegitimate state to a legitimate state is sometimes referred to as the convergence span. In essence, the properties of self-stabilization help in addressing many of the properties in self-management. A comparison of the two concepts is presented here.

- Self-configuration: consider the set of working configurations as legitimate states of the system. Self-stabilizing networks could self-configure themselves toward such a working configuration starting with an arbitrary configuration setup.

- Self-optimization: consider the set of optimal states of the system as the legitimate state set. As the environment changes, the system can find itself in a sub-optimal state, however, by the properties of self-stabilization, the system has a tendency to converge back to an optimal/legitimate state.
- Self-healing: if the system is to find itself in an illegitimate state due to faults, it can recover from such failures as long as they are transient.
- Self-protection: in the presence of attacks, self-stabilizing networks can return to legitimacy as long as none of their essential system components has been compromised or lost. The concept of self-stabilization does not deal with malicious changes to system behavior.

Moreover, self-stabilization addresses a fundamental property of a network. Because of the distributed and dynamic nature of autonomic networks, there lacks an assurance on stability of the global system state as the result of individual component behaviors. In fact, the frequency of state changes brought about by the components' self-managing behaviors could be detrimental to the stability of the system. The closure property of self-stabilization is highly useful in this aspect as it guarantees that a self-stabilizing network could remain in operational states regardless of individual component changes.

In the context of autonomic networks, we require a refinement to the classical self-stabilization concept. To understand the necessity of this refinement, we first make distinctions between *component-* and *system-wise* stability, and *dynamic* and *fixed-state* stability. In addressing stability issues in a large-scale networking context, a component refers to an individual entity in the network, such as a route, an overlay, and a P2P network. While, the system refers to the global environment (e.g., the Internet). Component-wise stability deals with the stability of the individual component under the closure and convergence properties, while system-wise stability is concerned with the closure and convergence properties of the global system. For example, P2P networks is a popular application on the Internet today. It makes sense to study the self-stabilizing properties of each P2P network (i.e., component-wise stability), and it is also important to analyze the stability of the Internet with respect to the interactions of the P2P networks. Similar comparisons could be drawn between an autonomic network and its self-managing components. Thus far, research on self-stabilization has been focused on component-wise stability, such as designing self-stabilizing protocols and systems. One could argue this problem treatment is sufficient because today's Internet is carefully planned and engineered to ensure system-wise stability. Nevertheless, given the autonomic networking vision as presented in Sect. 1, there is a definitive need for studying system-wise stability in the autonomic networking context.

The closure property of Dijkstra's self-stabilization concept is concerned with bounding an entity (system or component) within its nominal operations. It gives rise to what is essentially dynamic stability in that the entity is not required to stay in a particular state. In fact, the self-stabilizing token ring example Dijkstra has presented in his work exhibit dynamic stability. In contrast, fixed-state stability requires the entity to remain in a single stable state unless disturbed. For autonomic

networking, depending on the scenarios, a particular refinement of self-stabilization is desired:

- component-wise and dynamic stability: this would apply to most autonomic components. They have a finite number of legitimate states of operations (e.g., monitor, analyze, plan, and execute) and the component is not required to be “locked” in any particular state.
- component-wise and fixed-state stability: this would apply to a component that requires a persistent state of operation. For example, an end-to-end service path in a session is averse to changes (i.e., service disruptions).
- system-wise and dynamic stability: this would apply to many decentralized control mechanisms such as leader election and contention for wireless channel. As long as a deterministic number of active components in the system are generated at any given time, the system is considered stable.
- system-wise and fixed-state stability: this has strong relevance to resource provisioning scenarios. For example, when resources are provisioned to services in the global environment (e.g., for end-to-end QoS [quality of service] assurance or fault protection), it is important that there exists stable global system states where the services would not change their resource demands constantly.

Furthermore, our discussion of dynamic and fixed-state stability also clarifies a literary misconception between Dijkstra’s stability and general stability. Dijkstra’s stability is concerned with dynamic stability under legitimate states. Hence a “stable” state is equivalent to a legitimate state according to Dijkstra. General stability is concerned with “locked” in a particular state. In fact, such a “stable” state may not even be legitimate. For example, a deadlock scenario in distributed systems would be considered stable under general stability concept but not under Dijkstra’s self-stabilization. Our definition of fixed-state stability considers a “locked” state in the set of *legitimate* states and thus it is a stronger form of self-stabilization rather than a re-definition of general stability.

3.2 Self-Stabilization and Game Theory

Analyzing system behavior of autonomic networks is quite difficult, due to the dynamics of the problem and the scale of the operations. At the abstract, one could perceive an autonomic network to be composed of a large set of individual entities, interacting with the environment to achieve their own objectives. This description rather fits the profile of a game model under game theory, where the entities of the system are the players. Furthermore, the Nash equilibrium definition exhibits the closure and convergence properties of self-stabilization: (1) if the system is in a Nash equilibrium, it will stay in equilibrium unless perturbed by force (closure) and (2) if the system is not in a Nash equilibrium, it has a tendency to move toward a Nash equilibrium (convergence). Because game theory is rather extensive and broad in scope, in the following paragraphs, we first attempt to relate important

concepts in game theory to autonomic networking and discuss their relevance and applications. Then we focus on a particular type of games, called congestion games, that have been studied in the field of networking research. Finally, we summarize the application of game theory in networking research thus far. Since the purpose of this section is not to introduce game theory, interested readers are encouraged to seek introductory literature on game theory, such as [34].

3.2.1 Games in Autonomic Networks

In general, not all games have Nash equilibrium; hence, one major branch of game theory is to model a problem in a game form and to determine whether Nash equilibrium exist. A game in its normal form consists of three factors. The players are the individuals interacting among themselves and the environment. Each player has a set of actions it may choose to perform, called strategies. A utility function (also called payoff function) is a mapping of a player's strategy set to \mathfrak{R} . Thus, a system state is described as the combination of each player's strategy choice. This is referred to as a strategy profile. For purpose of evaluation, a system utility function could be used to map the strategy profile to \mathfrak{R} . Informally, Nash equilibrium is a strategy profile in which no player can improve its payoff by changing its strategy unilaterally. A pure Nash equilibrium is a strategy profile in which each player chooses its strategy deterministically (i.e., only one strategy is chosen always). A pure Nash equilibrium corresponds to fixed-state stability if the equilibrium is unique. A mixed Nash equilibrium is reached when each player assigns a probability to choose each of its possible strategies. Thus, the particular choice of strategy at any given time is stochastic. Therefore, a mixed Nash equilibrium is always a dynamic equilibrium. This statement is further ascertained when considering the uniqueness of equilibriums. When a game has a unique Nash equilibrium, not only it is guaranteed to arrive at that equilibrium (convergence), but also it has the tendency to stay in that equilibrium (closure). This is the reason why a pure Nash equilibrium with unique equilibrium corresponds to fixed-state stability. The stochastic nature of mixed strategy equilibrium precludes such a guarantee. When analyzing system behavior and stability of networks, pure Nash equilibrium is often sought after. For example, we would like the global routing behavior or resource provisioning mechanism to be fixed-state stable.

Although the strategies of a player are depicted as a set. Sometimes it is possible to describe this set with a continuous function, implying the set size is actually infinite. For example, if an application is to determine how to distribute its workload across a number of servers, there are infinite combinations of distributions among the servers (assuming the workload is \mathfrak{R} -splittable). While other games could only be described as a set of discrete strategies. For example, choosing a path to route packets. In general, continuous strategy games are easier to analyze since the utility of the player is also continuous. Thus, by analyzing the characteristic of the utility function it is possible to deduce the existence of Nash equilibriums. Unfortunately, in autonomic networking, much of the game models have discrete strategy sets.

The basic definition of Nash equilibrium assumes players are rational and noncooperative. Games that do not have pure Nash equilibrium may in fact have one when considering cooperative players. In fact, the system utility of a cooperative game is generally greater than its noncooperative counterpart. In autonomic networks, there are always certain degree of cooperation among players (e.g., an autonomic component may use the service of other components, two autonomic entities may agree to share their knowledge). However, what makes the modeling difficult is that in cooperative games, classical game theory is mostly concerned with full cooperation among all the players. When considering limited cooperation among subsets of players, the game analysis becomes complex and quickly intractable. Similar circumstance arises when considering the role of information in a game. A game where players have access to global knowledge (i.e. strategy set of other players and their payoffs) is called complete information game and is simple to solve. Games of limited information are much more difficult and are addressed in the area of Bayesian games. Thus far, very simple Bayesian games have been shown to be analyzable and it is a far cry from the complex and large scenarios presented in autonomic networks.

In essence, the ability to obtain timely and accurate information about the environment and the other players is critical to the traceability of a game. This fundamental requirement could only be supported by a knowledge management system. Thus, a system's ability to self-stabilize and its convergence span is heavily influenced by the effectiveness of the knowledge management system. In Sect. 2, we have discussed some of the challenges and issues in providing knowledge. And in this section, we see that a component requires the strategies and payoff function of other related components, some of which (e.g. the payoff function of another component) may only be reasoned about over time.

3.2.2 Congestion Games

It is nontrivial to find simple and applicable game models in the networking field, and fortunately one such game type exists, in the form of congestion games. Congestion games were first introduced by Rosenthal [37] and later formalized by Monderer and Shapley [30]. It is a class of games in which all players share a common pool of resources and each of the players utility is a decreasing function of the number of players using the same resource (thus the term congestion games). The general game model offered by congestion game fits a wide selection of network and system problem scenarios. Consider the following:

Scenario 1: a network supports a finite number of self-configuring overlays (*players*). Each overlay contain a set of overlay nodes each of which is supported by a underlying network resource (*resource*). The processing speed and response time of that overlay node is an increasing function depending on the number of other overlay nodes supported by the same network resource. The total response time across an overlay could be viewed as the sum of processing and response time obtained at each overlay node. Hence in choosing an overlay configuration (*strategy*), each overlay wants to minimize the total response time.

Scenario 2: Consider a set of self-optimizing service routes in the network (*players*). Each route aims at choosing an optimal network path (*strategy*) that could minimize its end-to-end delay, which is represented as the sum of the delays over each network link along the path. The delay at a network link (*resource*) is an increasing function depending on the number of service routes it serves.

With global knowledge and the presence of a central planner, the solutions to the above scenarios could be obtained readily. However, in a self-managing network context, the players make their own decisions under the condition of non-cooperation and with limited information. There is no guarantee that the overall system has any stable states, and if so, whether convergence to a stable state is bounded. Both of the above scenarios could be described by a congestion game model:

Let $\Gamma_D = \langle N, \{Y_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$ be a game in strategic form. N is the finite set of players $\{1, \dots, n\}$, Y_i is the finite set of strategies available to player i and $u_i : Y \rightarrow \Re_+$ where $Y = Y_1 \times Y_2 \dots \times Y_n$ is the payoff function of player i . Given a finite set of resources $T = \{t_1, \dots, t_m\}$, define $Y_i \subset 2^T$. Let $A_i \in Y_i$ be a strategy of player i , $A \in Y$ be a strategy profile, c_j be the cost function of resource t_j , and l_j be the normalized serving capacity of t_j , then

$$u_i(A) = \sum_{j \in A_i} c_j(A)$$

$$c_j(A) = \frac{x_j(A)}{l_j}$$

$$x_j(A) = \#\{i \in N : t_j \in A_i\}$$

Γ_D is a multicommodity game. Unlike single-commodity models, a multicommodity game does not limit the players from choosing more than one resource in a single strategy. We observe that the cost function c_j of resource t_j is a strictly increasing function solely depending on the number of players using t_j . Finally, the game is asymmetrical since each player may have very different strategy set (e.g., different overlays may be configured over different segments of the network, and different service routes may have different source and destination pairs).

A congestion game is called a potential game when there exists a system potential function such that the increase in utility of a player incidentally causes a drop in potential. Hence, all potential games have at least one Nash equilibrium (i.e., the global minimal of the system potential). Thus, congestion games that have potentials are highly valuable to obtain self-stabilization in autonomic networks. Fortunately, the congestion game model we have presented here in fact has a potential. Secondly, the establishment of potential serves as the basis for distributed decision making without need for global knowledge. The fact that a player's self-utility improvement naturally leads to the minimization of system potential alleviates the lack of global information in games. Lastly, the definition of congestion at each resource greatly simplifies information gathering for each player. For each player, rather than the

need for knowing other player's individual actions in order to compute the utility, the aggregate measure of congestion level at the resources is sufficient. Furthermore, such information is readily obtainable from the network today.

3.2.3 Related Works

Significant amount of works have been conducted on network problems based on congestion game models. Some are able to establish the existence of pure Nash equilibria in congestion games and to determine their complexity (e.g., [9, 13, 29]). In general, there is no guarantee that a pure Nash equilibrium exists in all congestion games [29], and when it does, the number of steps it takes for the system to converge is exponential in worst case [13]. Furthermore, the models used in studying convergence are often over simplified and hence difficult to apply in practice. For instance, the popular K-P model [22] is a single-commodity model that assumes all players have a common source and destination, choose a single resource from a shared resource collection, and the resources are independent (e.g., parallel links).

In multicommodity congestion games with simultaneous moves, whether convergence to a pure Nash equilibrium could be bounded is still an open question and examples could be found in which convergence does not occur. Because of its complexity, the study of convergence in general congestion games has been mainly focused on finding convergence bound to approximate solutions. Christodoulou, Mirrokni, and Sidiropoulos [7] bounded the solution after one round of best-response walk by all players to $\Theta(n)$ -approximate in general case. Chien and Sinclair [6] showed that when the increase in cost of adding a player is bounded ("bounded jump" condition), convergence to Nash equilibrium occurs in polynomial time. Goemans, Mirrokni, and Vetta [14] studied convergence of Nash dynamics to "sink equilibrium," which is not an approximate of a pure Nash equilibrium. In fact, a sink equilibrium could be formed by a group of cyclic states in some cases. This in fact corresponds to dynamic stability.

Game theoretical analysis has been conducted in many fields of network research in the past (e.g., pricing, flow control, route stability, efficiency of wireless networks). Some works have examined the existence of unique Nash equilibrium in non-cooperative user-based routing environments [2, 33]. Orda, Rom, and Shimkin [33] have shown that in two-node multilink network topology, there exists an unique Nash equilibrium. For general networks, the uniqueness of Nash equilibrium is guaranteed if the cost functions of links have diagonal strict convexity, the users share the same source and destination and are symmetric in cost functions, or each user assigns positive flows to all the links in the network. Altman et al. [2] studied noncooperative routing games under general topology network with polynomial cost function. They have shown the uniqueness of Nash equilibrium under bounded cost. Computation of the Nash equilibrium is carried out as user-based global optimization problem where users have the same source and destination, across parallel links and each user assigns positive amount of load on each link in the network. A special form of routing game termed bottleneck routing game is investigated by Banner and Orda [5]. In such a game, the user attempts to minimize the load of its bottleneck

link, rather than to minimize the end-to-end cost. The existence of Nash equilibrium is shown and for unsplittable flows, polynomial time convergence bound is obtained.

3.3 Other Theories Supporting Self-Stabilization

Price of anarchy [36] is often studied in bounding the optimality of a Nash equilibrium. It arises because game theory studies focus on stability while the traditional network and system planning focus on performance. Thus it follows that there should be some evaluation on the performance of Nash equilibrium. Price of anarchy is often denoted as the ratio between the system utility of a Nash equilibrium and the system optimal (e.g., obtained via centralized planning).

Emergence from biology describes how simple local behaviors by entities without global knowledge result in coordinated global behavior. In [3], the emergence concept is used to design an “emergent” election algorithm. A key feature of emergence is that interactions are generally between the components and the environment (e.g., emitting pheromone trails), and the communications are one way and independent. Individual messages in emergent systems have low values on their own and the system behavior is nondeterministic. These features of emergence make their designs extremely simple on the component level and render component validation tractable. Thus far, it is uncertain whether emergent designs could lead to complex and yet stable system behaviors in self-managing networks. Another related concept is stigmergy: insects coordinate their behaviors by using environmental modifications as cue. Work in collaborative construction [45] gives promising lessons in this direction, where swarms of robots are able to construct complex building structures from blocks by following simple rules and observe local environmental stimuli. The similarities between the collaborative construction and the self-organizing properties of autonomic networks suggest that indeed it is possible to find design problems, such as distributed clustering and election, in networks using stigmergy concepts.

4 Conclusion

In this chapter, we have examined two essential and not well-addressed issues in autonomic networks: knowledge management and self-stabilization. Through an in-depth discussion of their concepts and applications in autonomic networks, we hope the readers have gained a feel for the importance of these problems and an understanding of existing approaches to address them.

Knowledge management is an emerging research area that aims to provide a unified infrastructure for managing network knowledge. Compared to traditional network management, not only knowledge management is responsible for storing and retrieving network information, it is also responsible for correlating and reasoning about the information to construct a consistent view of the system. As future communication networks will get even more complex in scale and technology,

knowledge management will become a crucial support for autonomic networking. Although many key issues of network knowledge management have been partially or starting to be addressed, much work is still required to bring network knowledge management to maturity.

Stabilization in autonomic networking is a difficult problem due to the lack of centralized control and the scale of the networks. In drawing connection with Dijkstra's self-stabilization theory and research on game theory, we have shown that indeed it is feasible to address the problem in formal and analytical ways, especially through the study of congestion games. Furthermore, it is critical to map the autonomic networking scenarios into simple yet effective game models for the resulting analysis to be meaningful and trackable. As discussed, currently there is still a rather large gap between what the research could tell us and what is needed for practical self-stabilizing network/system designs.

Together, knowledge management and self-stabilization fills in two much needed gaps in autonomic networking. While knowledge management is the foundation for all autonomic behaviors (including self-stabilization), self-stabilization provides the necessary management and control at the global system level, despite underlying distributed and often self-motivated behaviors of the individual components.

References

1. M. Alavi and D. Leidner, in *Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues*, MIS Quarterly, vol. 25, no. 1, pp. 107–136, JSTOR, 2001.
2. E. Altman, T. Basar, T. Jimenez and N. Shimkin, in *Competitive Routing in Networks With Polynomial Cost*, IEEE INFOCOM, 2000.
3. R. Anthony, in *Emergence: a Paradigm for Robust and Scalable Distributed Applications*, IEEE 1st International Conference on Autonomic Computing, May 2004.
4. H. Arora, B. Mishra and T. Raghun, in *Autonomic-Computing Approach to Secure Knowledge Management: A Game-Theoretic Analysis*, IEEE Transactions on Systems, Man and Cybernetics, Part A, vol. 36, no. 3, pp. 107–136, JSTOR, 2001.
5. R. Banner and A. Orda, in *Bottleneck Routing Games in Communication Networks*, IEEE INFOCOM, April 2006.
6. S. Chien and A. Sinclair, in *Convergence to Approximate Nash Equilibria in Congestion Games*, ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.
7. G. Christodoulou, V. Mirrokni and A. Sidiropoulos, in *Convergence and approximation in potential games*, Symposium on Theoretical Aspects in Computer Science (STACS), 2006.
8. D. Clark, C. Partridge, C. Ramming and J. Wroclawski, in *A Knowledge Plane for the Internet*, ACM SIGCOMM, 2003.
9. C. Daskalakis, P. Goldberg and C. Papadimitriou, in *The Complexity of Computing a Nash Equilibrium*, 38th ACM Symposium on Theory of Computing (STOC), 2006.
10. E.W. Dijkstra, in *Self-Stabilization In Spite of Distributed Control*, Communications of the ACM, vol. 17, no. 11, pp. 643–644, 1974.
11. S. Dobson, S. Denazis, A. Fernandez, D. Gaiti, E. Gelenbe, F. Massacci, N. Paddy, F. Saffre, N. Schmidt and F. Zambonelli, in *A Survey of Autonomic Communications*, ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 1, no. 2, pp. 223–259, 2006.
12. S. Dolev, in *Self-Stabilization*, The MIT Press, March 2000.

13. A. Fabbrikant, C. Papadimitriou and K. Talwar, in *The Complexity of Pure Nash Equilibria*, 36th ACM Symposium on Theory of Computing (STOC), 2004.
14. M. Goemans, V. Mirrokni and A. Vetta, in *Sink Equilibria and Convergence*, 46th IEEE Symposium on Foundations of Computer Science (FOCS), 2005.
15. Y. Gong, M. Lu, G. Wang and K. Zhou, in *Research on Process Knowledge Management Based on Ontology*, Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2007.
16. M. Hellerstein, A. Jain, S. Ratnasamy and D. Wetherall, in *A Wakeup Call for Internet Monitoring System: The Case for Distributed Triggers*, Proceedings of the Third Workshop on Hot Topics in Networks (HotNets-III), 2004.
17. R. Huebsche, J. Hellerstein, N. Lanham, B. Loo, S. Shenker and I. Stoica, in *Querying the Internet With PIER*, Proceedings of the International Conference on Very Large Data Bases (VLDB), 2003.
18. IBM, in *Autonomic Computing Architecture: A Blueprint for Managing Complex Computing Environments*, IBM and Autonomic Computing, October 2002.
19. B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. Foghlu, W. Donnelly and J. Strassner, in *Towards Autonomic Management of Communication Networks*, IEEE Communications Magazine, vol. 45, no. 10, pp. 112–121, 2007.
20. Y. Kalfoglou, T. Menzies, K. Althoff and E. Motta, in *Meta-Knowledge in Systems Design: Pannacea or Undelivered Promise?*, The Knowledge Engineering Review, vol. 15, no. 4, pp. 381–404, 2000.
21. J. Kenney, D. Lewis, D. O’Sullivan, A. Roelens, V. Wade, A. Boran and R. Richardson, in *Runtime Semantic Interoperability for Gathering Ontology-based Network Context*, Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2006.
22. E. Koutsoupias and C. Papadimitriou, in *Worst-Case Equilibria*, Symposium on Theoretical Aspects in Computer Science (STACS), 1999.
23. D. Lewis, J. Kenney, D. O’Sullivan and S. Guo, in *Towards a Managed Extensible Control Plane for Knowledge-Based Networking*, Lecture Notes in Computer Science, vol. 4269/2006, 2006.
24. J. Lopez de Vergara, V. Villagra, J. Asensio and J. Berrocal, in *Ontologies: Giving Semantics to Network Management Models*, IEEE Network, vol. 17, no. 3, pp. 15–21, 2003.
25. J. Lopez de Vergara, V. Villagra and J. Berrocal, in *Applying the Web Ontology Language to Management Information*, IEEE Communications Magazine, vol. 42, no. 7, pp. 68–74, 2004.
26. H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson and A. Krishnamurthy, in *iPlane: An Information Plane for Distributed Services*, Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI), 2006.
27. M. May, M. Siekkinen, V. Goebel, T. Plagemann and R. Chaparadza, in *Monitoring as First Class Citizen in an Autonomic Network Universe*, Proceedings of the Workshop on Technologies for Situated and Autonomic Communications (SAC), 2007.
28. T. Menzies, K. Althoff, Y. Kalfoglou and E. Motta, in *Issues with Meta-Knowledge*, International Journal on Software Engineering and Knowledge Engineering (SEKE), vol. 10, no. 4, pp. 549–555, 2000.
29. I. Milchtaich, in *Congestion Games With Player-Specific Payoff Functions*, Games and Economic Behavior, vol. 13, pp. 111–124, 1996.
30. D. Monderer and L. Shapley, in *Potential Games*, Games and Economics Behavior, vol. 14, pp. 124–143, 1996.
31. R. Mortier and E. Kiciman, in *Autonomic Network Management: Some Pragmatic Considerations*, Proceedings of the SIGCOMM Workshop on Internet Network Management (INM’06), September 2006.
32. M. Mulvenna, F. Zambonelli, K. Curran and C. Nugent, in *Knowledge Networks*, Lecture Notes in Computer Science, vol. 3854/2006, 2006.

33. A. Orda, R. Rom and N. Shimkin, in *Competitive Routing in Multiuser Communication Networks*, IEEE/ACM Transactions on Networking, vol. 1, no. 5, pp. 510–522, October 1993.
34. M. Osborne and A. Rubinstein, in *A Course in Game Theory*, The MIT Press, July 1994.
35. A. Peddemors, I. Niemegeers, H. Eertink and J. de Heer, in *A System Perspective on Cognition for Autonomic Computing and Communication*, Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA), 2005.
36. C. Papadimitriou, in *Algorithms, Games and the Internet*, 33rd ACM Symposium on Theory of Computing (STOC), 2001.
37. R. Rosenthal, in *A Class of Games Possessing Pure-Strategy Nash Equilibria*, International Journal of Game Theory, vol. 2, pp. 65–67, 1973.
38. M. Schneider, in *Self-Stabilization*, ACM Computing Surveys, vol. 25, no. 1, p.45-67, 1993.
39. Smaha, in *Haystack: an Intrusion Detection System*, Aerospace Computer Security Applications Conference, 1988.
40. M. Smith, C. Welty and D. McGuinness, Eds., in *OWL Web Ontology Language Overview*, Available online at <http://www.w3.org/TR/owl-features/>, 1990.
41. R. Sterritt and M. Hinchey, in *Why Computer-Based Systems Should be Autonomic*, IEEE 12th International Conference and Workshops on the Engineering of Computer-Based Systems, April 2005.
42. J. Strassner, in *Knowledge Management Issues for Autonomic Systems*, Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA), 2005.
43. R. Van Renesse, K. Birman and W. Vogels, in *Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management and Data Mining*, ACM Transactions on Computer Systems, vol. 21, no. 2, pp. 164–206, 2003.
44. M. Wawrzoniak, L. Peterson and T. Roscoe, in *Sophia: An Information Plane for Networked Systems*, ACM SIGCOMM Computer Communication Review, vol. 34, no. 1, pp. 15–20, 2004.
45. J. Werfel and R. Nagpal, in *Extended Stigmergy in Collective Construction*, IEEE Intelligent Systems, vol. 21, no. 2, pp. 20–28, 2006.
46. P. Yalagandula and M. Dahlin, in *A Scalable Distributed Information Management System*, ACM SIGCOMM, 2004.

Autonomic Networking in Wireless Sensor Networks

Mengjie Yu, Hala Mokhtar, and Madjid Merabti

Abstract In this chapter, we address autonomic networking in terms of wireless sensor networks (WSNs), a typical example of wireless networks in pervasive computing. In order to investigate the state of the art of autonomic networking in sensor networks and its future prospects, we start with a short summary of autonomic networking and Sensor networks. It follows the discussion of the appliance of autonomic networking in WSNs and existing research approaches. After that, we adopt fault management as an example to demonstrate how autonomic networking and architecture-based approaches fits into the design of autonomic sensor networks.

1 Introduction

Autonomic networking, derived from the *self* concepts of autonomic computing by IBM [14] in 2003, has been gradually recognized as an efficient paradigm to design robust self-managing computing systems in large-scale distributed networks. The ultimate goal of such networking is to relieve the management burdens of computing systems from human administrators, and have systems manage themselves spontaneously according to unpredicted events or changes in harsh operational environments. Such system self-governs its behaviours and maintains its performance. It usually interacts with human administrators at the policy level, for example taking the high-level system objectives. Significant features of autonomic networking include self-optimization, self-diagnosis, self-configuration, self-healing and self-adaptation; nevertheless, self-awareness of the system operational status and surrounding environment is an essential requirement to design system autonomic behaviours. In general, the autonomy of networking systems is achieved through the collaboration of a set of self-managed entities instead of the traditional centralized control. Self-managed entities collaborate together to adapt systems' performance promptly and accurately to unpredicted changes in the operational environment. For

M. Yu (✉)

School of Computing and Mathematical Science, Liverpool John Moores University, Byrom Street, Liverpool, UK

e-mail: M.Yu@2001.ljmu.ac.uk

example, self-managed entities plan the actions, and integrate required resources to monitor the task through to completion with less or without the manual configuration and intervene from human administrators. As a result, human is eligible to focus more on the business logic design rather than addressing the details of low-level system integration and configuration.

The success of autonomic networking fundamentally relies on the appropriate localisation of the management processes within the network systems. Recent autonomic technologies vary in forms of agent-based approaches, swarming algorithms or emergent biologically inspired scenarios. In this section, we examine the state of the art of autonomic networking management and its techniques in wireless sensor networks, a typical example of wireless network in pervasive computing.

Before we describe the operation of autonomic networking and its technologies in WSNs, it will be helpful if we summarize several unique features in relation to the design of robust wireless sensor networking systems as in Sect. 1.1. Also, we discuss the appliance of autonomic networking in sensor networks in Sect. 1.2. Sect. 1.3 examines the existing autonomic networking approaches in sensor networks. Finally, we conclude the importance and necessity of applying architecture-based approach to facilitate the autonomy of sensor networks in Sect. 1.4.

1.1 Wireless Sensor Networks

Wireless sensor network (WSN) [1] was initially inspired by the request of military usage to monitor and collect information in the battlefield, such as enemy movements, explosions and other phenomenon of interests in the battlefield. Sensor networks are hence adopted to facilitate the development of new wireless monitoring and controlling environmental applications [20], such as health applications in hospitals, home automation and smart environments, or traffic control systems. For example, extracting useful information such as temperature, sound or pollution.

Recent advances in micro-electro-mechanical system technologies, wireless communications, and digital electronics have boosted the vision of deploying large numbers of miniaturized electronic devices to monitor harsh operational and nature environments. Based on the node collaborative efforts, sensor networks provide enterprise applications access to *real-word* information by collecting, processing and disseminating sensor data from operational environments. They are expected to operate for periods of time ranging from days to years.

As in Fig. 1, a sensor node usually consists of a small processing unit with limited computational power (CPU) and memory, a communication device including radio transceivers, multifunctional sensor units and power source in form of a button cell battery. The small dimension design of sensor nodes also place strong restrictions on their hardware and software capabilities in terms of processing capabilities, memory storage, energy supply and so on. This eventually influences the networking and system design in sensor networks. In this section, we address couple of unique characteristics in relation to the design of efficient wireless sensor networks.

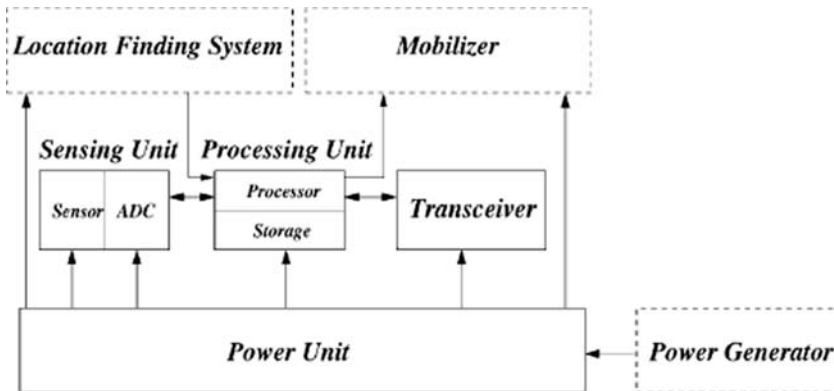


Fig. 1 Wireless sensor networks structure

1.1.1 Critical Operational and Deployment Environments

Sensor nodes are usually deployed very close or directly inside the monitored phenomenon. They might operate in the dense forest, in military battlefield [17], on the ocean bed or attached to moving objects (such as animals, vehicles), etc. Unpredicted events (e.g. node failure caused by nature conditions such as fire in forest) and harsh operational environments (e.g. heat as in vehicle engine, pressure at the bottom of ocean) bring extremely critical challenges on both software and hardware performance of sensor nodes. Therefore, applications of sensor networks are expected to be robust and adaptive to these unforeseen events occurred in the network.

1.1.2 Network Scalability

The number of sensor nodes in WSNs is various according to the sensing application requirements on the targeted phenomenon, such as the accuracy and quantity of sensor data and the reliability of fault tolerance on node failure. The density of sensor nodes in a region or the network might range from hundreds to thousands [41], or even reaching an extreme value of millions. It is thus unfeasible for human administrators to manually fix the malfunction or faulty sensor nodes, especially in area where the presence of human is impracticable. Networking management systems of WSNs are likely expected to 'self'-respond to various events and reassume the network performance from node failure.

1.1.3 Hardware Constraints

Miniaturization technology allows sensor nodes to be designed in low cost and small dimension, and deployed in large numbers. It also restrains the performance and capabilities of both software and hardware of sensor nodes. For example, the processing unit, associated with a small storage unit, is only feasible to support

sensor nodes handling small amount of data and simple tasks (i.e. data integration). It is unfeasible to implement complex networking management functions in sensor nodes as the traditional distributed approach does. Another significant feature is that sensor nodes are usually powered by battery unit such as button cell. The lifetime of a sensor network largely relies on the battery expenditure of sensor nodes. As a result, the shortage and depletion of node battery become one important fact that affects the consistence and efficiency of sensor networking performance. Note that the battery consumption of the local computational cost of a node incurs more energy efficient than the message radio transmission (including both sending and receiving) in sensor networks [21]. Thus, sensor nodes are encouraged to lessen the radio transmission for energy conservation.

1.1.4 Network Topology

For all these reasons discussed previously, sensor network topology is expected to change frequently according to the operational environment conditions and node resource expenditure status. To reassume the network connectivity and sensing coverage in a region, it demands the collaboration of sensor nodes to self-reorganize and configure the sensor networks spontaneously to the changes.

1.2 Autonomic Networking in Wireless Sensor Networks

Given the large scale of network and unfeasible physical accessibility, it is extremely difficult for human administrators to manually manage and configure sensor networks. For example, identifying the faulty nodes in the phenomenon area; replacing the button cell battery of sensor nodes and reconfiguring network topology in a region where the node density might range from hundreds to thousands. To achieve these, sensor networks are expected to be self-managed and self-adaptable dynamically to various changes in the operational environment.

It is unfeasible to rely on few 'smart' networking components to achieve the self-managed feature in sensor networks as it does in the traditional wired networks. The inherent resource constrains present obstacles for sensor nodes to execute the complex functionality and sophisticatedly consult with the central controller for handling the dynamic networking events. 'Selfness' of sensor networks has to fundamentally rely on the collaboration of sensor nodes and their individual 'self'-behaviours. Thus, the autonomy aspects of autonomic networking can be well fitted into the design requirements of autonomic sensor networks in a holistic manner.

The perspective of autonomic networking motivates the proposition that sensor networks are self-managed by interacting human administrators only at the policy level. The autonomy of sensor networks is achieved through the self-configuration of a node as individual component behaviour and self-organization of a set of nodes as the collaboration manner.

Self-configuration: Sensor nodes reconfigure and adapt their behaviours of networking and sensing by altering parameter values dynamically to the changing

conditions and states of the network. For example, decreasing the node's sensing duty cycle if the monitored phenomenon has no significant changes in a period of time; and reducing radio transmission power to shorten the communication range if the residual node energy has dropped to a critical level. These autonomic behaviours support nodes to energy-efficiently manage and conserve their valuable battery, which eventually prolong the lifetime of the entire network.

Self-organization: Sensor nodes collaborate to reconfigure the network performance within a region or the entire network. For example, to lessen the impact of faulty nodes, sensor nodes surrounding the phenomenon area generally regroup among themselves in order to maintain the reliability and consistence of network connectivity and sensing coverage.

1.3 Existing Autonomic Networking Issues in Sensor Networks

In this section, we examine the existing approaches of autonomic networking in wireless sensor networks. These approaches vary in forms of node power management, routing protocol, network topology management and fault management. Note that the entire sensor network lifetime usually relies on the existence of individual sensor nodes. The more energy-efficient node performance is, the longer lifetime of sensor networks can be prolonged. As a result, energy efficiency has been widely considered as an important and essential fact by these autonomic approaches.

1.3.1 Power Management

The lifetime of WSN relies on the battery lifespan of sensor nodes. To conserve the network energy, adjusting node activities [38] according to system requirements and node hardware status has been widely considered as one effective solution. For example, decreasing hardware operating frequency when node is not in the peak performance state; lessening the sampling frequency of sensor node if the risk of missing a crucial event of monitored phenomenon is tolerant and switching some nodes into 'sleeping' model if there are enough nodes to monitor phenomenon.

To distinguish node activities, Sinha [30] defined three operational models for the processing unit of a sensor node as active, idle and sleep mode. Based on instantaneous data processing requirements and identifying the peak performance, a node adopts different operational models for its processing unit. This is achieved by adjusting the power supply and operating frequency. Thus, node power consumption can be reduced when the processing unit is in either idle or sleep mode. Another efficient technique is to switch off redundant nodes and temporarily put them into 'sleep' or 'idle' model. Cerpa [2] proposed an approach to keep a small number of sensor nodes in active while idling redundant nodes for energy conservation. To properly identify redundant nodes in the network, Richard [35] adopted a mathematic-based interpolation technique. If the interpolation error of a sensor node is less than a pre-defined application threshold, this node is thus hibernated for conserving further amounts of battery energy. By periodically assessing the redundant

nodes and putting them in idle, all nodes in the network are guaranteed to spend some portion of their operational time in a low power mode. This consequently reduces the overall power consumption of sensor networks.

The radio transceiver is the most power-hungry hardware unit of a sensor node. Therefore, it is desirable to lessen node radio transmission and reception activities if necessary. At an autonomic level, Marsh [19] enabled nodes to adjust their sampling rates adapting to the changes of monitored phenomenon. For example, if sensor reading becomes more reliable in a period of time, node reduces its sampling rate (e.g. from every 0.5 s to every 8 s) and becomes less frequent to transmit data back to the base station for energy conservation. Moreover, sensor nodes are demanded to adjust their radio transmission ranges dynamically to changing conditions or state of the network. It is achieved by altering the node transmission power. For example, in MAANA [29], autonomic sensor nodes execute self-configuration service to change their communication range according to their distance from the application Access Point (AP).

1.3.2 Routing Management

To lessen unnecessary radio transmission and conserve network energy, there is a need to design energy-efficient routing protocols for sensor nodes to transmit data packet. Instead of flooding messages over the network and demand nodes to transfer redundant data, certain autonomic wisdom supports sensor nodes to selectively execute data transmission. One technique is the data-centric and content-based routing schema. Sensor nodes identify and respond to application query announcement that they are only interested. As a result, parts of network conserve energy without involving data transmission. Typical examples of such are SPIN [26], direct diffusion [13], etc.

Nodes along the routing path that is used frequently for data transmission might have the rapid energy depletion rate than other nodes. To balance the network energy and prevent sudden network partition, advanced self-managed capability is introduced to enable nodes optimally choose the best path due to the residual energy and the state of the network. For example, Gan [8] proposed a routing protocol that supports nodes to choose the next transmission hop based on the assessment of energy cost among potential routing paths. It also takes in consideration of the remaining energy of neighbouring nodes. This ensures data traffic is evenly spread over the network. Therefore, nodes conserve energy, and the network survivability consequently increases. In addition, monitoring data transmission misbehaving of neighbouring nodes is another technique to select the next hop-node in the routing path. Marti [22] required sensor nodes to constantly monitor whether data packets have been transmitted properly by their neighbours. Node is thus capable of choosing a shortest and optimal path that contains most reliable intermediate nodes.

To balance nodes' energy depletion in the network, Heinzelman [11] scheduled sensor nodes to send data back to sensing applications in turns. The elected node aggregates sensor data from others, and adopts a long-range transmission power to

route data directly to the base station. This requires no intermediate nodes, and the amount of data transmitted in the network is also reduced.

1.3.3 Network Topology Management

Network topology management is one important property and feature of wireless sensor networking. The topology usually affects many networking characteristics such as system latency, robustness and capacity.

Given the facts of large-scale deployment and potential placement in harsh environment, it is essential for sensor network to be self-organized and self-reconfigurable. Moreover, sensor nodes are highly vulnerable to the harsh operational environment, and prone to fail. Nodes might also fail because of their resource depletion such as limited battery energy. These unforeseen facts unexpectedly cause the network partition, and threaten the consistence of networking performance in WSNs. Therefore, the network must be able to reconfigure itself periodically based on the collaboration of sensor nodes. In particular, Heinzelman [11] proposed an energy-efficient adaptive clustering technique to dynamically manage the network topology. It enables sensor nodes to periodically self-organize and coordinate to form groups (in terms of cluster) in the network. Sensor nodes are elected as the cluster heads in turns to evenly share the energy load such as the energy cost of managing clusters and communicating with the base station. This design balances the energy consumption in the network, and avoids the rapid energy depletion of certain nodes as in the static topology model. In addition, Smaragdakis [31] proposed their topology algorithm to elect cluster-heads based on weighting the residual energy of sensor nodes in the network. Moreover, Younis [42] furthered this scenario by taking into consideration of a secondary parameter as the node's proximity to its neighbours or node degree.

1.3.4 Fault Management

We particularly adopt fault management as an example in Sect. 3 to demonstrate how autonomic networking fits into the design of efficient wireless sensor networks.

1.4 Autonomoic Management Architecture for Sensor Networks

The success of node autonomy in sensor networks fundamentally relies on the appropriate localization of the management processes within the network systems. One technique is architecture-based approach. Architecture-based approach are expected to improve sensor network performance with low-overhead of sensor nodes. It lessens the radio communication in the network, and coordinates nodes' self-behaviours dynamically to various changes in the environments. The details of autonomic architectures in sensor networks are reviewed in Sect. 2.

2 Architectures for Autonomic Networking in Sensor Networks

In this section, we investigate the state of the art of management architectures that facilitates autonomic networking in sensor networks. The use of architecture is a good strategy to deal with complex networking management in resource-constrained wireless sensor networks. Efficient architecture is eligible to energy-efficiently distribute management processes among sensor nodes. Sensor nodes are thus self-manageable, and require few consultancies with the central manager in the network. This lessens the in-network radio communication, and conserves node battery energy to prolong the lifetime of network. Sensor networks are usually deployed in the highly dynamically changing environment. Robust architecture allows autonomic nodes to be aware of the surrounding environment, and respond spontaneously to events occurred in the network. Thus, sensor networks can be self-managed based on the node collaboration efforts, with little or no human intervene. We first explore what autonomic components are in sensor networks. We then discuss the existing architecture approaches, and classify them into different catalogues.

2.1 *Autonomic Components in Sensor Networks*

The vision of autonomic sensor networks relies on the autonomy of sensor nodes and their collaboration. Sensor nodes are expected to equip with sufficient functions and knowledge to handle situations occurred in the network, even those that were unforeseen in future. Given a task and objective, autonomic nodes create a plan for actions, integrate required hardware and software resources and complete management tasks.

One technique is to apply intelligent software agent. Agent-based approach fulfils the basic requirements of autonomic networking, such as function distribution, artificial intelligence and self-management. Its appliance to sensor networks results into various reasons.

First, agent brings sensor nodes the autonomy to self-manage and self-optimize in the remote area on behalf of human administrator or sensing applications. Unlike the traditional client–server management model, agent-based systems do not always maintain a link with the central controller. This design is extremely useful in sensor networks, as it lessens in-network communication and thus conserves nodes' energy.

Second, agent allows nodes to be aware of its environment, and react accordingly without sophistic consultancies from central management system. This reduces the response delay towards unpredicted events occurred in the network.

Third, flexibility is one major benefit of agent appliance in sensor networks. The concept of swappable and reconfigurable software components of agents has drawn much attention to design robust networking functions in autonomic sensor networks. It decomposes complex management functionality into smaller sub-processes. It allows resource-constrained sensor nodes to selectively choose essential software

components to maintain their networking tasks; while, the agent-based functions of sensor nodes are also able to alter and easily update according to the new requirements.

Based on the deployment location and functionality of agent, autonomic nodes are various in forms of their management structure as manager-agent (MA), mobile-agent and multi-agent approaches.

2.1.1 Manager-Agent

MA model is one of the most common structures in sensor networks. It distributes networking functions among sensor nodes based on their management roles. Manager resides on the node that holds and maintains a globe view of a large set of sensor nodes. It executes a range of autonomic management services, including coverage area maintenance, topology management and failure detection service. To handle complex management services, it usually selects the base station node (powered by wired supply) or nodes (with sufficient resources such as power energy) as manager. Meanwhile, agents act as the representation of manager to monitor and manage nodes in the remote region. For example, in MANNA [27], manager resides on the base station node. It is responsible for manage and monitor the entire network by processing complex functions. It also generates information models including network topology and energy model based on the information collected from the sensor nodes. Agents are assigned to nodes that are normally more powerful than the common nodes in both hardware and software capabilities. These nodes are elected to monitor a set of common nodes in the remote region on behalf of the manager. They are responsible to collect nodes status and execute local networking management functions, and perform aggregation of management data retrieved from common nodes. These nodes adjust their radio communication range and directly forward management data to the base station. This avoids using intermediate nodes for energy conservation. Based on the collected network information, manager is also eligible to command agents to execute management services on common nodes such as failure detection and idling redundant nodes. As a result, networks can manage themselves without direct and manual human configuration for a period of time.

2.1.2 Multi-Agents

Multi-agent system (MAS) provides a robust platform to support the collaboration and negotiation of autonomic nodes in the network. Unlike the 'Manager and Agent' model, there is no manager role in MAS. Agents spread all over the sensor nodes, and cooperate together to complete management tasks such as identifying and idling redundant nodes for energy conservation in the monitored phenomenon area. These agents are much stronger than the one in MA model. The notion of strong derives from the fact that such agent has explicit beliefs, desires and intentions about its behaviours and the environment surrounding its activities. They usually do not consult with any central managing controller. They hold the beliefs (pre-defined

criteria) to assess their present activities and surrounding environment conditions. They have plans to response to events occurred in the network, and adopt steps to complete their management goal. They might re-evaluate their belief and knowledge as new information and requirement becomes available. For example, consultation with high-level autonomic management systems to update their networking management goals.

O'Hare [25] presented a good example of how multi-agents coordinate autonomic nodes on their power management. They consider a node might irregularly and unexpectedly fail and disconnect from the network because of its battery depletion. This results into the sudden network partition, which might affect the network connectivity and sensing coverage of monitored phenomenon area. To prevent it and last the network performance as long as possible, it is necessary to lessen the activities of that node and, meanwhile, keep it remain active to ensure the network connection. Power management agents of nodes collaborate to handle the conflicting task by balancing the needs of network connection and coverage with the residual nodes' status. Negotiation between nodes take place to identify which node is eligible to shift extra workload and willing to take on the extra energy drain to routing the data towards the base station. The elected node is thus demanded to adjust its activities, such as increasing its sampling rate and transmission frequency towards the base station.

Agents of sensor nodes might not always need to negotiate among themselves to handle the networking management tasks. Marsh [19] demonstrated a intruder detection model that relies on the individual autonomy of a node to alter its behaviour based on the results gained from the data processing and a set of pre-defined logical rules. Such node automatically reduces its data transmission frequency for energy conservation when the monitored intruder events become less regularly. This autonomic behaviour bases on the knowledge that node must keep it active as long as possible but also must report any unforeseen unusual events. Compared to the non-agent design, it offers a great reduction in energy depletion for data transmission, and consequently prolongs the network lifetime. To update autonomic behaviours, agent's knowledge of nodes is easily altered via mobile agent if additional functionality and requirement arises. The details of mobile agent and its usage are discussed in Sect. 2.1.3.

Furthermore, multiple agents are also allowed to reside on a single sensor node to enhance its autonomic behaviours. Multiple agents collaborate among themselves to handle the complex networking management, or address a small problem individually. This design decomposes the complex management tasks into various small sub-processes, tailored to the computational constraints typically for nodes in sensor networks. In particular, Agilla [7] proposed a system architecture to coordinate the autonomy of multi-agents on a node used in a fire tracking application. In this application, Agilla provides a tuplespace for the reliable coordination of agents within a node, or even remote access between two nodes. The tuplespace servers as the communication platform between agents on a node, or among nodes. Agilla also adopts mobile agent technique to deploy new requirements and additional functionality throughout a sensor network.

2.1.3 Mobile Agent

Mobile agents have been widely adopted by existing research approaches to design energy-efficient systems in sensor networks. It is considered to greatly reduce the communication cost of sensor nodes, especially over networks designed with low bandwidth radio links. In general, there are two major usages of mobile agent applying in sensor networks: (1) disseminating sensor data efficiently from nodes to the base station and (2) updating autonomic nodes with additional knowledge or functionality when new requirements occur.

Data Dissemination

In the traditional client/server-based sensor networks, sensor data is directly transferred to the base station via multi-hop communication after nodes collect from the monitored phenomenon area. The base station thus processes and integrates those data. Whereas, in the mobile-agent-based paradigm, it transfers the processing function to the destination area, integrates sensor data and only sends the result back to the base station instead of forwarding all the unprocessed data. This greatly reduces the number of data packets transferred in the network, and consequently improves the network performance in terms of node energy consumption and the packet delivery ratio. In particular, Chen [4] proposed a mobile-agent-based architecture to design energy-efficient sensor nodes by reducing and aggregating data within the network. The base station dynamically deploys the executable processing codes (in terms of mobile agent) into the target area based on the requirement of a specific application. Mobile agent migrates to the nodes in the remote area, and locally processes the raw data at the source nodes. Only the aggregated results are sent back to the base station. This capability enables the reduction in the amount of data transmission by allowing only relevant information to be extracted and transmitted in the network. Consequently, it conserves the node energy, and prolongs the lifetime of sensor networks. Moreover, Wang [37] also adopted mobile agents to aggregate and process raw sensor data in the remote area. However, in this design, mobile agents also migrated back to the base station with the aggregated data results. The localized data aggregation process helps mobile agent reduce the data transmitted in the network, and thus it saves network bandwidth and node energy.

Functionality and Knowledge Update

Mobile agents are transferred to the destination area to complete specific management tasks on behalf of the base station. When the new challenge and requirement occurs, agents of nodes within that area might have not equipped with relevant functionality and knowledge. The base station injects a mobile agent containing the feasible functions into the network, routing towards the region of interest. The mobile agent is thus executed in the destination nodes on behalf of the base station. For example, in O'Hare [25] power management approach, mobile agent is adopted to support the base station on assessing whether there is a feasibility to reduce the

sampling rate of destination nodes for energy conservation. The mobile agent is dispatched to the destination node. It acts the representative of the base station to assess the redundancy situation including node's status, and the changes of monitored events. The decision is thus made at local computational process rather than having all relevant data streamed back to the base station for analysis. In addition, Gan [8] proposed mobile agents to assist aggregated sensor data from source nodes routing back to the base station by selecting the optimal path. As we have discussed previously in Sect. 1.3.2, Gan considered the data routing traffic should be spread over the network rather than using the same optimal path frequently. Using the same path depletes the battery energy of nodes along that path rapidly than other nodes, and in the worst case may lead to the network sudden partition. To do that, mobile agent travels with the aggregated data together towards the base station. Specifically, when it reaches an intermediate node, it obtains the local information of the current node and its neighbouring nodes. Based on the information, it assists a node to choose the next routing hop by taking into consideration of the energy cost of potential routing paths and resident energy of nodes in the network. After arriving at the base station, it passes the data and then dies.

The context of mobile agent is usually various depending on the new requirements from sensing applications or human administrator. As a result, sensor network has the capability to alter its performance (e.g. node autonomic networking behaviours) on demand.

2.2 Existing Architecture for Autonomic Sensor Networks

Efficient architecture usually presents a sufficient communication paradigm for autonomic nodes to dynamically collaborate to handle the complex networking management tasks. It also describes the distribution of networking management functionality among sensor nodes (such as manager, agents). Architecture aims to improve the node collaboration with low-overhead through lessening the in-network radio communication. So far, couples of networking management architectures have already been developed for autonomic sensor networks. In general, these architectures can be classified according to their management structure as flat, distributed and hierarchical model.

2.2.1 Flat Management Architecture

Flat management architecture is characterized by sensor nodes that have the same hardware and software capabilities in the network. Agents usually reside within sensor nodes, and bring their autonomy to locally handle the network management tasks. In order to avoid the large amount of traffic generated in the network, nodes do not have to maintain a sophistic consultancy with the central manager. They make their own decision adapting to events occurred in the network. The manager, residing on the base station, is responsible to monitor the globe view of the entire network based on the information collected from sensor nodes. When new requirements

occur, the manager dispatches mobile agents to carry the on-demand functionality and knowledge towards nodes in the destination area. The typical examples of flat management architecture are found in [4, 8, 19, 25, 37], etc.

2.2.2 Distributed Management Architecture

It splits the networking management of the entire network into several sub-regions. Each region elects a node to server as the manager (usually in forms of cluster-head). The cluster-head is responsible for the networking management in its own sub-region on behalf of the base station. One typical example is the ‘Manager-Agent’ based model as in MANNA [27] discussed previously. To support cluster-heads in a cooperative fashion, it also proposed a simple network management protocol called MannNMP [29] to exchange management information. Moreover, Tai [34] adopted cluster-based approach to distribute fault detection services among different regions. Cluster-head detects the suspicious nodes among its member nodes in one-hop communication range. This reduces a large number of management messages routing back to the base station, and consequently conserves the network energy. To provide the communication between neighbouring clusters, a gateway node, which is in one-hop range of both two cluster-heads, propagate fault management reports across the network. In these approaches, cluster-heads are usually elected among nodes with powerful extra capabilities (in terms of communication, energy and computation, etc.) to execute complex management tasks. In addition, Heinzelman [11] adopted an approach that rotates cluster-head role over time among different nodes to balance the resource consumption within a cluster. They believed each node has the same capabilities in both hardware and software. All nodes can transmit data with enough power to reach the base station if needed. Cluster-head manages its member nodes and periodically collect management information from them. It is also responsible to route the aggregated data of its cluster directly to the base station. This avoids the intermediate nodes to forward the data packets, and consequently reduces the in-network communication.

2.2.3 Hierarchical Management Architecture

In the hierarchical model, management tasks are mainly taken by the cluster-heads nodes. Cluster-heads are responsible to manage the member nodes in their own region, and usually do not communicate and coordinate with each other. They only maintain the link with the base station to disseminate management information and take management objectives. This approach normally applies in the heterogeneous networks, in which the cluster-heads are elected from nodes with extra resource capabilities. Therefore, such nodes are carefully placed in the network to contribute to the management performance of the sensing and networking. In particular, Niu [24] proposed a three-tier hierarchical management structure to detect suspicious nodes in the networking. Clustering is adopted to lessen the in-network communication, and group sensor nodes around the phenomenon area to detect the suspicious events. They believed transferring sensor measurement back to the base

station via multi-hop communication has very high signal-to-noise ratio. While, clustering surrounding the phenomenon area can accurately detect the target events in the short communication distance. Cluster-heads make decisions about the fault events occurred in their sub-regions, and further transferred the reports to inform the base station. Moreover, to accomplish networking management in a reliable and energy-efficient manner, Ying [40] proposed a mobile-agent-based hierarchical policy management architecture for WSNs. Policy Manager (PM) resides in the base station at the highest level; Cluster Policy Agent (CPA) sits in the node with best resources in a cluster; Local Policy Agent (LPA) manages a sensor node and also enforce local networking management tasks such as analysing network dynamics (e.g. topology change), performing configuration, monitoring and reporting. They adopted the mobile agents the transfer and update the management policy when users want to execute a specific task in the network. Policies are thus propagated from the PM to CPAs to LPAs, or from CPAs to LPAs. As a result, it keeps the management consistence without halting the networking processes for manual reconfiguration. It also increases the adaptability and re-configurability of the networking management systems.

3 Fault Management in Autonomic Sensor Networks

In this section, we adopt fault management as an example to demonstrate the autonomic behaviours of sensor networks on handling various unforeseen events such as node failure or malfunction.

The intention of fault management for WSNs is different from traditional wired networks. For example, we might be interested in the results of fault management within a given region (or overall network) rather than simply fixing an individual node failure or an individual wireless connection between two nodes. Existing fault management approaches for WSNs vary in forms, such as architectures [16, 28, 34], protocols [22, 26], detection algorithms [3, 6, 12] and detection decision fusion algorithms [5, 36]. This section starts from fault detection, and follows the discussion through fault diagnosis and failure recovery.

3.1 Fault Detection

Fault detection is the first phase of fault management, where an unexpected failure should be properly identified by the network system. The traditional centralized approach has represented its inefficiency in handling large-scale sensor networks. The distinctive problem is that the central node easily becomes a single point of data traffic concentration for fault detection and fault management in the network. This subsequently causes a high volume of message traffic and rapid energy depletion in certain regions of the network, especially the nodes closer to the central node. Extra burden is incurred forwarding the communication messages from other

nodes. In addition, the multi-hop communication of this approach will also increase the response delay from the base station towards faults in the network. Therefore, efforts have been made to seek a distributed and more computationally efficient fault detection model. So far, the autonomy of sensor nodes has been widely considered as an efficient solution. The basic idea behind it is to enable a sensor node to make decisions at certain levels. The more decisions a node can make, the less information (i.e. the number of messages) needs to be delivered to the central node. As a result, this approach conserves the node energy and consequently prolongs the network lifetime. Examples of such development are node fault self-detection and self-correction on its hardware physical malfunction (i.e. sensor, battery, RF transceiver) [10], failure detection via neighbour coordination [3, 6, 12], utilization of WATCHDOG to detect misbehaving neighbours [22] and distributed fault detection by use of cluster technology [34]. Others address the use of decision fusion centres to make the final decisions about suspicious nodes [5, 36].

3.1.1 Neighbour Coordination

Neighbour coordination is one example of fault management distribution. Nodes coordinate together to detect suspicious nodes before consulting with the central controller (e.g. the base station). Usually, the central controller is not aware of any failure unless it is believed to be wrong with high confidence by the neighbouring nodes. This design reduces the in-network communication traffic, and subsequently conserves node energy. For example, suspicious (or failed) nodes are identified via comparing their sensor readings with neighbours' median readings. With this motivation, Ding [6] developed a localized algorithm to identify suspicious nodes whose sensor readings have large differences against their neighbours. Although this algorithm works for large-scale sensor networks, the probability of sensor faults needs to be small. If half or more of the sensor neighbours are faulty, the algorithm cannot detect the faults as efficiently as expected. However, Chen [3] have improved this approach to identify suspicious nodes even when half of a node's neighbours are faulty. It chooses the good node in the network, and uses its best sensor readings to diagnose others' status. This information can be further propagated through the entire network to diagnose all other sensors as being either good or faulty. Neighbour coordination is also applied to enhance the accuracy of failure detection. Instead of sending out an alarm right after a fault is detected, a node is expected to consult with its neighbour for concluding a more accuracy decision. One approach is the two-phase neighbour coordination scheme, as in Hsin [12]. A node waits for its neighbours to update information concerning the suspicious node in the first phase. In the second phase, it consults with its neighbours to reach a more accurate decision. As a result, the monitoring effect is propagated across the network. The central node only needs to monitor a potentially very small subset of sensor nodes. However, this approach requires the network to be pre-configured. Each sensor node should have a unique ID, and the central node knows the existence and ID of each node. In addition, special software components are also applied to support neighbour coordination. WATCHDOG [22] – as provided by the routing

layer of a node – is used to detect failed or misbehaving neighbours if data packets have not been transmitted properly. However, this process may be slow, and is error-prone because the resource-constrained sensor nodes cannot be expected to constantly police all of their neighbours. Consequently, it may end up routing into a new neighbour that has also failed.

3.1.2 Clustering Approach

Clustering is an emerging technology to distribute fault management tasks in wireless sensor networks. It sets up a virtual communication skeleton to group nodes and splits the overall network into different groups (e.g. clusters). Fault management is distributed and executed in each individual group. Usually, the leader node of a cluster (e.g. the cluster-head) executes fault detection in its group via a centralized approach. It detects the suspicious nodes by exchanging the heartbeat messages with its group members. A pre-defined failure detection rule is applied in the cluster head to identify the failed nodes. Moreover, in Ann Tai [34], the gateway node (GW as shown in Fig. 2) is also adopted to propagate the local detected failure information to all other clusters in the network. A Gateway node is considered as the neighbour of the cluster heads of two different clusters. This approach makes local clusters aware of the changes and management objectives of the overall network.

3.1.3 Distributed Detection

The fundamental idea of distributed detection is to have nodes make decision on faults such as physical malfunction of sensing devices. A node is expected to self-monitor or self-detect faults with the less consultation from the central controller. As one approach, special hardware and software components are applied to detect physical impact on the sensor node. For example, Harte [10] adopt a flexible circuit using accelerometers that act as a sensing layer around a node to detect malfunctions of the node. They also use software components (e.g. ADCC, TimerC) from the TinyOS operating system of a node to sample and detect the misbehaviour of sensor devices.

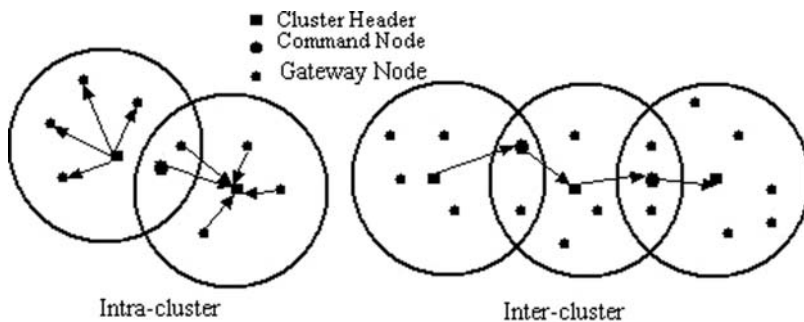


Fig. 2 Intra-cluster heartbeat diffusion and inter-cluster information propagation [34]

Alternatively, the decision of fault detection may also be generated by a node (or several) after aggregating decision from nodes in the network. A typical example is shown in Fig. 3, where a fusion node detects abnormal sensor readings of a node after comparing and aggregating data from a set of nodes. The number of fusion sensors needed in the network usually depends on the area of the region of interest and the communication range of individual sensor nodes. Note that this approach is especially energy-efficient and ideal for data-centric sensor applications. However, there remain various research challenges in order to achieve a better balance between fault detection accuracy and the energy usage of the network. Usually, the efficiency of such failure detection schemes is counted in terms of node communication costs, precision, detection accuracy and the number of faulty sensor nodes tolerable in the network. One of the techniques suggested is fusion sensor coordination. In Clouqueur's work [5], fusion sensors (in terms of manager nodes) coordinate with each other to guarantee that they obtain the same global information about the network before making a decision, as faulty nodes may send them inconsistent information. In addition, Wang [36] also consider a fault-tolerant solution to reduce the overall computation time and memory requirements at the fusion sensors. This approach adopts cluster technology for data aggregation and lessening of redundant data.

The distributed approach has represented a major design trends for fault management in WSNs. Sensor nodes are gradually taking more management responsibility and decision-making in order to fulfil the vision of self-managed WSNs. Node self-detection schemes [10] and neighbour coordination [12] have provided us with good examples of fault management distribution. However, they only focus

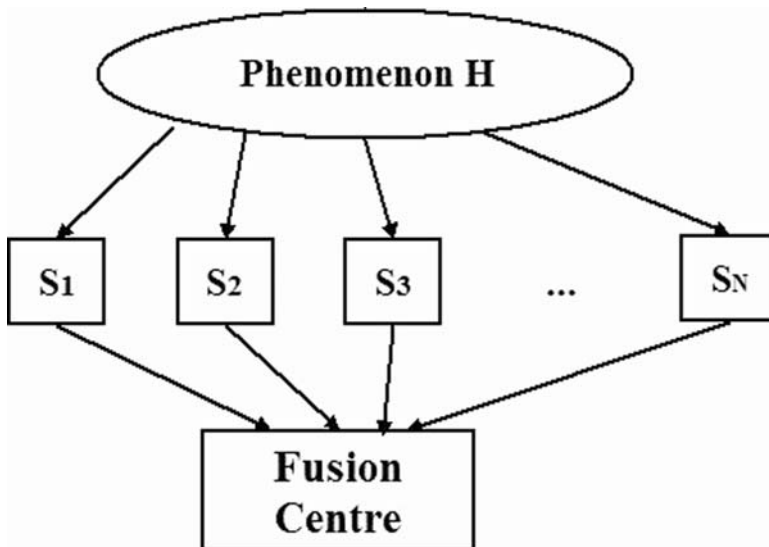


Fig. 3 Distributed event detection with fusion centre [5]

on small regions (a group of nodes) or individual nodes. Research work such as MANNA [28] and WinMS [16] has proposed management architectures able to look after the overall network using a central manager scheme. As an example of a scheme that draws on the benefit of both techniques, we introduce our own hierarchical management structure [39], which assigns levels of management activities (or responsibilities) to nodes in the network. This approach does not fully rely on the centralized manager for analysis and decision-making. We consider the essence of fault management functions to be the same for all nodes, but differing in terms of the execution ranges (e.g. number of nodes) and degrees (e.g. execution frequency and sequence of functions). As a result, the centralized manager concentrates on the performance of the overall network, without needing to be particularly interested in the state of any individual node, while the individual group manager looks after nodes within its sub-region.

3.2 Fault Diagnosis

As the second stage of fault management, detected faults are properly identified by the network system and also distinguished from the other irrelevant or spurious alarms.

The accuracy and correctness of fault detection has already been partly addressed and achieved using a number of fault detection methods [5, 12, 34]. However, there is still no comprehensive description model to distinguish various faults in WSNs. To support the autonomy of sensor networks in achieving accurate fault diagnosis or fault recovery action, the efficient knowledge model including fault description models will become a necessity in the future perspectives of research challenges. Existing approaches address fault models from the individual node point of view (including node hardware component malfunctions). In particular, both Chen [3] and Koushanfar [15] assume that the system software has already been fault-tolerant. They focus on the sensor node hardware faults, especially sensor and actuator faults which are most prone to malfunctioning. Koushanfar adopted two fault models. The first one relates to sensors that produce binary outputs. The second fault model is related to the sensors with continuous (analog) or multilevel digital outputs. Clouqueur [5] only consider faulty nodes due to the harsh environmental conditions. In their work, faulty nodes are assumed to send inconsistent and arbitrary values to other nodes during the information sharing phase. Ding [6] models the ‘event’ (i.e. abnormal behaviour of a sensor node) using the real numbers (such as sensor readings) instead of using the 0/1 (binary) decision model. As a result, their algorithm is generic enough as long as the thresholds and real number of events can be specified by fault tolerance requirements from various sensor applications.

To enhance to the accuracy and correctness of fault diagnosis, techniques such as using knowledge sharing and ‘histrionic’ data have been applied to identify suspicious nodes from failure alarms. For example, Gaurav [9] encourages the gateway nodes (i.e. the head of a group of nodes) to construct an overall network

communication map by sharing their own connectivity experience with other gateway nodes. If none of the gateways has received the status update from a certain gateway node, it clearly indicates that the node is not able to transmit any data to others due to its transmitter failure. Using ‘histrionic’ data is another typical example to accurately identify the faults in sensor networks. Shoubhik [23] demonstrated an approach of using a priori knowledge model of sensor data to predict and identify errors in the received data packet. This knowledge model is used to predict future samples of sensed data based on the limited amount of past records. The success of this approach thus depends largely on the accuracy of the modelling of the sensor data. As a result, the sink node is able to choose the most faultless routing path to deliver data packets by computing the average prediction error over the routing paths towards the destination. Moreover, in order to provide sufficient runtime data for fault diagnosis, multiple copies of sensed data from a source node are transmitted to the sink via the separate paths. For example, Kuo-Feng et al. [33] proposed an approach that a source node forwards two copies of the data through different routing paths towards the sink node. If these two received copies are not identical, the sink node assumes a data inconsistency failure may have occurred in the network. It thus requires the source node to retransmit the data via a third separate path. The sink node examines the contents of these data packets, and identifies the faulty paths (and nodes) in the network. To identify the suspicious node in the faulty path, a sink node sends diffusion data packet back to the source node through both correct and faulty paths. Every time when data packet passes through the suspicious node, a counter embedded in the data packets will increased. When the counter reaches a pre-defined threshold setting, the node is regarded as faulty.

As we have mentioned previously, existing fault models either address at the component level (e.g. the hardware and software) of sensor nodes, or the communication connectivity between nodes. Note that there is a need to address fault models from the network and system management level (e.g. network connectivity, the rate of failed nodes).

3.3 Fault Recovery

Ideally, failure recovery phase is the stage at which the sensor network is restructured or reconfigured. In such a way, failures or faulty nodes do not impact further on the network performance. Most existing approaches in sensor networks isolate failed (or malfunction) nodes directly from the network communication layer (e.g. the routing layer). For example, in the approach of Marti [22], after the faulty neighbour is detected, a node will choose a new neighbour to route to. Staddon [32] proposed two approaches of resuming the network routing paths from the silent nodes (i.e. failed nodes), which are detected in each network routing update epoch. Both Kuo-Feng [33] and Shoubhik [23] suggested the sink node to select a new routing path when the faulty path occurred in the network. Additionally, to recover the network connectivity from the node leader’s failure (e.g. cluster header, gateway node), the

group members usually rejoin into the adjacent groups. Gaurav [9] proposed an runtime recovery mechanism to enable the members of the failed gateway node to reconnect to the network. If a node is in the same communication range of multiple candidate groups, it is recommended to join the group with the minimum communication energy cost. Instead of only responding to the fault occurred in the network, the proactive action is considered as a novel approach to prevent the potential faults in the future. In WinMS [16], the central manager detects the network region with the weak health (e.g. low battery power) by comparing the current network status (including individual nodes) with a historical network information model (e.g. an energy map or topology map). It takes the proactive action by enabling nodes in that area to send data less frequently for the energy conservation. Koushanfar [15] suggested a back-up scheme for tolerating and healing the hardware malfunctions of a sensor node. They believe a single type of hardware resource can back up different types of resources. The key idea is to adapt application algorithms and/or operating system to match the available hardware and applications needs. They currently focus on five primary types of resource: computing, storage, communication, sensing and actuating, which can replace each other via suitable changes in system and application software. Although this solution is not directly related to fault recovery in respect of the network system management, it still provides us with a useful vision of a future design to reconfigure the node management functionality when its management responsibility has changed due to fault occurred in the network. Currently, one of the obvious ways to update the management functionality of a node is the use of mobile code technology [18]. However, we believe using different combinations of the existing software components of a sensor node as a means to reconfigure its management capability is an alternative solution against the use of mobile code technology. This may provide a more efficient solution when node management responsibility changes.

4 Future Challenges

The unique characteristics of wireless sensor networks have made the networking management approach different from the traditional ones in wired networks. It poses additional technical challenges such as network discovery, network routing, data processing and management task coordination among resource-constrained sensor nodes. Moreover, the unforeseen events (e.g. node fault because of resource depletion or nature condition in environment) make sensor networks even vulnerable to failure. This consequently results into the sudden network partition, and threatens the consistence of sensor network in both networking and sensing performance. To avoid that, the system autonomy becomes a necessary and desirable feature for sensor networks to dynamically handle the complex networking management tasks, especially in area where the presence of human administrator is impracticable.

The novel feature of autonomic sensor networking fundamentally relies on the individual autonomy of sensor nodes and their efficient collaboration. So far,

agent-based approach has been widely adopted to equip nodes with relevant knowledge and functionality, and even update with the additional capabilities if necessary. However, an agent is normally designed and assigned with application-specific tasks, and processes in destination nodes on behalf of the high-level manager. It prevents the obstacle for agents to handle the complex networking-related management tasks. For example, the network topology issue which usually considers many networking characteristics such as system latency, data routing and sensor coverage. This demands autonomic nodes have sufficient knowledge to determine common causes of problems through real-time analysis of residual network measurements. Moreover, there is also a need for autonomic nodes to be self-aware of its own status and the environment surrounding its activities. Thus, nodes are eligible to define and schedule the contexts for the autonomic control adapting to various events occurred in the network. To achieve this, the robust and efficient system architecture of sensor nodes becomes a new technical research challenge in sensor networks. System architecture, sitting between the sensing applications and node operational system (such as TinyOS), is expected to provide a set of integrated functions for nodes to be self-manageable and self-configurable. Nevertheless, there is still no comprehensive system architecture design than supports the autonomy of sensor networking. Design challenges for system architecture are various in different perspectives. In this section, we adopt fault management as an example to highlight couple of design aspects of system architecture in sensor networks.

First, system architecture is demanded to have sufficient knowledge model. As we have discussed previously, it is unfeasible to predict any node failure, in particular the future, under the harsh operational environment of sensing applications and WSNs. To support the management system to take efficient recovery actions and successfully resume from a failure, certain knowledge (or wisdom) is required to clearly identify and distinguish various faults. Therefore, the development of theoretical and realistic knowledge models of various faults becomes a key technique and requirement. As reviewed in Sect. 3, several existing approaches have already addressed this on the individual node level in terms of sense data inconsistency, network connectivity failure, and node hardware malfunction. Note that there is a need to address fault models not just at the level of components and individual nodes, but also at the network and system management level.

Second, there is always a trade-off between the complexity of fault management functions of system architecture and the resource constraints of sensor nodes. Modularization of fault management functions have gradually been considered as an efficient design for resource-constrained sensor networks. In addition, this approach eases the function maintenance as small changes (e.g. code updates and reconfiguration) within a selected section of system architecture will not affect the whole architecture.

Third, reconfiguration of fault management functionality of sensor node is one major challenges of system architecture. As mentioned above, mobile code update technology [18] has been proposed as an efficient scheme. It selectively chooses and updates the parts of code that are needed to support adaptation and reconfiguration of node functionality. In order to achieve this, sensor nodes in the network

are required to distribute and forward the on-demand mobile code to the destination nodes. However, we are seeking an alternative technique [39] based on the reconfiguration of existing management functions by altering action ranges and degrees of a sensor node. In association with our hierarchical structure, execution frequency and the sequence of functions generates management functionality for a node when its management responsibility changes. The techniques we are proposing are still under development, so it would be grossly premature to suggest we provide a solution to all the difficulties involved. At the very least, we hope to show that we can apply new perspectives to these research problems.

5 Conclusion

In this chapter, we address autonomic networking in terms of WSNs, a typical example of wireless networks in pervasive computing. We investigate the state of the art of autonomic networking in sensor networks and its future prospects. We also emphasize the importance and efficiency of the appliance of architecture to design autonomic sensor networks. After that, we adopt fault management as an example to demonstrate how autonomic networking and autonomic architectures fits into the design for robust sensor networks.

Acknowledgments Our research group would like to express gratitude to Engineering and Physical Science Research Council (EPSRC). This research project is sponsored by EPSRC under its grant reference EP/D000092/1. We would also like to thank the special issue editors for their patience and comments during the reviewing of this paper.

References

1. Akyildiz, I.F.: Wireless Sensor Networks: A Survey In: IEEE Computer Networks, 38, pp. 392–422. IEEE (2001)
2. Cerpa, A.: ASCENT: Adaptive Self-Configuring Sensor Networks Topologies. In: INFOCOM'02, pp. 1278–1287, New York, USA. IEEE (2002)
3. Chen, J.: Distributed Fault Detection of Wireless Sensor Networks. In: DIWANS'06, pp. 65–72, Los Angeles, CA, USA, ACM Press (2006)
4. Chen, M.: Mobile Agent Based Wireless Sensor Networks. In: Journal of Computers, pp. 14–21, 1(1), (2006)
5. Clouqueur, T.: Fault Tolerance in Collaborative Sensor Networks for Target Detection. In: IEEE Transactions on Computers, 53(3), pp. 320–333. IEEE (2004)
6. Ding, M.: Localized Fault-Tolerant Event Boundary Detection in Sensor Networks. In: INFOCOM'05, pp. 902–913, Miami, USA. IEEE (2005)
7. Fok, C.L.: Mobile Agent Middleware for Sensor Networks: An Application Case Study. In: IPSN'05, pp. 382–387, Los Angeles, USA. IEEE (2005)
8. Gan, L.: Agent-Based, Energy Efficient Routing in Sensor Networks. In: AAMAS'04, New York, USA, ACM Press (2004)
9. Gupta, G.: Fault-Tolerant Clustering of Wireless Sensor Networks.. In: WCNC'03, pp. 1579–1584, New Orleans, USA. IEEE (2003)

10. Harte, S.: Fault Tolerance In Sensor Networks using Self-Diagnosing Sensor Nodes. In: intelligent environments conference, IEE (2005)
11. Heinzelman, B.W.: An Application-Specific Protocol Architecture for Wireless Microsensor Networks. In: IEEE Transactions on Wireless Communications, 1(4), pp. 660–669. IEEE (2002)
12. Hsin, C.: Self-monitoring of Wireless Sensor Networks. In: Computer Communications, 29(2006), pp. 462–478, IEEE (2005)
13. Intanagonwiwat, C.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: MOBICOM '00, Boston, USA, ACM Press (2000)
14. O.Kephart, J.: The Vision of Autonomic Computing. In: Journal of IEEE Computer, 36(1), pp. 41–50, IEEE (2003)
15. Koushanfar, F.: Fault Tolerance Techniques for Wireless Ad Hoc Sensor Networks. In: IEEE Sensors conference, pp. 1491–1496, Orlando, USA, IEEE (2002)
16. Lee, W.L.: WinMS: Wireless Sensor Network-Management System, An Adaptive Policy-Based Management for Wireless Sensor Networks. In: The University of Western Australia, Technical Report Number: UWA-CSSE-06-001 (2006)
17. Maroti, M.: Shooter Localization in Urban Terrian. In: IEEE Computer, 37, pp. 60–61, IEEE (2004)
18. Marron, P.J.: Management and Configuration Issues for Sensor Networks. In: Journal of Network Management, 15(4), pp. 235–253, IEEE (2005)
19. Marsh, D.: Autonomic Wireless Sensor Networks. In: Engineering Applications of Artificial Intelligence, pp. 741–748 (2004)
20. Martinez, K.: Environmental Sensor Networks. In: IEEE Computer, 37, pp. 50–56, IEEE (2004)
21. Mathur, G.: Ultra-Low Power Data Storage for Sensor Networks. In: IPSN'06, pp. 374–381, Nashville, USA, IEEE (2006)
22. Marti, S.: Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. In: MOBICOM'00, Boston, USA, ACM Press (2000)
23. Mukhopadhyay, S.: Model Based Error Correction for Wireless Sensor Networks. In: SECON 2004, pp. 575–584, Santa Clara, CA, IEEE (2004)
24. Niu, R.: Distributed Detection and Fusion in a Large Wireless Sensor Network of Random Size. In: Journal on Wireless Communications and Networking, pp. 462–472, IEEE (2005)
25. O'Hare, G.M.P.: Agents for wireless sensor network power management. In: ICCP'05, pp. 413–418, IEEE (2005)
26. Perrig, A.: SPINS: Security Protocols for Sensor Networks. In: MobiCom'01, Rome, Italy, ACM Press (2001)
27. Ruiz, L.B.: MANNA: A Management Architecture for Wireless Sensor Networks. In: IEEE Communications Magazine, 41(2), pp. 116–125. IEEE (2002)
28. Ruiz, L.B.: Fault Management in Event-Driven Wireless Sensor Networks. In: MSWiM'04 , pp. 149–156, Venice, Italy. ACM Press (2004)
29. Ruiz, L.B.: On the Design of a Self-Managed Wireless Sensor Network. In: IEEE Communication Magazine , 43(8), pp. 96–102. IEEE (2005)
30. Sinha, A.: Dynamic Power Management in Wireless Sensor Networks. In: IEEE Design and Test of Computers, 18(2), pp. 62–74, IEEE (2001)
31. Smaragdakis, G.: SEP: A Stable Election Protocol for Clustered Heterogeneous Wireless Sensor Networks. In: SANPA'04, Boston, USA (2004)
32. Staddon, J.: Efficient Tracing of Failed Nodes in Sensor Networks. In: WSN'A'02, Atlanta, USA, ACM Press (2002)
33. Su, K.F.: Detection and Diagnosis of data inconsistency failures in wireless sensor networks. In: Journal of Computer Networks, 50(9), pp. 1247–1260. IEEE (2006)
34. Tai, A.T.: Cluster-Based Failure Detection Service for Large-Scale Ad Hoc Wireless Network Applications in Dependable Systems and Networks. In: Dependable Systems and Networks, IEEE (2004)

35. Tynan, R.: Intelligent Agents for Wireless Sensor Networks. In: IA-WSN'06, Hong Kong, P.R.China, IEEE (2006)
36. Wang, T.Y.: Distributed Fault-Tolerant Classification in Wireless Sensor Networks. In: Journal on Selected Areas in Communications, 23(4), pp. 724–734. IEEE (2005)
37. Wang, X.L.: Collaborative Multi-Modality Target Classification in distributed Sensor Networks. In: Information Fusion'02, 1, pp. 285–290. IEEE (2002)
38. Ye, F.: PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks. In: ICNP'02, pp. 200–201, Paris, France, IEEE (2002)
39. Yu, M.: A Self-Organized Middleware Architecture for Wireless Sensor Network Management. In: Journal of Ad Hoc and Ubiquitous Computing, pp. 135–145, Inderscience (2008)
40. Ying, Z.: Mobile Agent-based Policy Management for Wireless Sensor Networks. In: WCNM'05, 2, pp. 1207–1210, Wuhan, China, IEEE (2005)
41. Yoneki, E.: A Survey of Wireless Sensor Network Technologies: Research Trends and Middleware's Role. In: Technical Report, 646, University of Cambridge, p. 45, Cambridge (2005)
42. Younis, M.: Architecture for Efficient Monitoring and Management of Sensor Networks. In: Lecture Notes in Computer Science, vol: 2839/2003 pp. 488–502, Springer (2003)

iNetLab: A Model-Driven Development and Performance Engineering Environment for Autonomic Network Applications

Hiroshi Wada, Chonho Lee, Junichi Suzuki, and Tetsuo Otani

Abstract A key software engineering challenge in autonomic computing is the complexity of administrating operational policies of applications. In order to address this challenge, this chapter proposes and evaluates a new development environment, called iNetLab, which is designed to improve the productivity of designing, maintaining, and tuning operational policies in autonomic network applications. iNetLab consists of (1) a set of visual modeling languages specialized to define operational policies in network applications, (2) a set of supporting facilities for those modeling languages, and (3) tools estimates the performance of a network application with its operational policy under development. The proposed visual modeling languages and their supporting facilities can simplify and semi-automate the process to design and maintain operational policies by allowing application administrators (i.e., non-programmers) to graphically deal with operational policies in an intuitive manner. The proposed performance estimation tools leverage the performance history of each network application (i.e., pairs of an operational policy and a performance result obtained in the past) and approximate the application's performance without deploying and running it actually. This simplifies the process to tune operational policies against desirable performance requirements and contributes to shorten the time to develop autonomic applications.

1 Research Issues in Autonomic Computing

A key software engineering challenge in autonomic computing is the complexity of managing operational policies, each of which defines an administrative decision (i.e., a pair of an operational condition and an administrative action) to operate applications. This complexity derives from two major issues: (1) information overloading in designing and maintaining operational policies and (2) a lack of guidance methods to tune operational policies against desirable performance requirements such as response time, throughput and load balancing.

H. Wada (✉)
University of Massachusetts, Boston, MA, USA
e-mail: shu@cs.umb.edu

The first issue is information overloading in designing and maintaining operational policies. It is often tedious, expensive, and error-prone for application administrators to design proper operational policies for their applications because policy design involves a large volume and variety of information [14]. Administrators need to consider a variety of operational conditions (e.g., each application component's internal conditions such as resource utilization and external conditions such as workload) and administrative actions (e.g., migration of an application component from one host to another). Due to this information overloading, administrators can be easily overwhelmed to pair operational conditions and administrative actions as operational policies. This problem becomes even more serious when an application's functional and/or non-functional aspects often change. Functional changes include introducing new application functionalities and updating existing ones. Non-functional changes include adding new hardware resources and revising service level agreements. Upon these changes, administrators need to consider additional operational conditions and administrative actions and re-design operational policies by re-pairing them again.

The second issue is a lack of guidance methods to tune operational policies against desirable performance requirements [15, 18]. Although each operational policy defines a set of administration actions for application components to take under certain operational conditions, it is nothing to do with the performance that they can collectively yield with its operational policy. It is always hard to estimate whether a given operational policy allows an application to satisfy particular performance requirements. As a result, a number of operational policies need to be evaluated extensively with a simulator or testbed in order to determine which one to be used in an application. This trial-and-error evaluation process can take a significant amount of time and costs. Also, it is often too ad hoc and unreliable to guide application administrators to obtain a reasonable operational policy that can satisfy desirable performance requirements.

2 Key Components and Contributions of iNetLab

In order to address the two issues described in the previous section, this chapter proposes and evaluates a new development environment, called iNetLab, which is intended to improve the productivity of designing, maintaining, and tuning operational policies in autonomic network applications. iNetLab consists of three key components described below.

The first component in iNetLab is a set of visual modeling languages that are specialized to design and maintain operational policies (i.e., operational conditions and administrative actions) in network applications. They are intended to be visual domain-specific languages (DSLs) that directly capture, represent, and implement domain-specific concepts; the concepts specific to operational policies in autonomic network applications, rather than general-purpose languages that aim at general software problems. The proposed DSLs define those domain-specific concepts in their metamodels and represent them as visual or textual language primitives. This

simplifies the process to build visual models that allow application administrators (nonprogrammers) to intuitively understand, design and maintain operational policies. Moreover, the proposed DSLs intentionally limit their expressiveness to specify operational policies only; therefore, they can reduce the chances for application administrators to make errors by building models in invalid or unexpected ways. By simplifying the design of operational policies and automating their validation, the proposed DSLs reduce the cost to design and maintain operational policies, thereby alleviating the issue of information overloading.

The second component in iNetLab is a set of supporting facilities for the proposed DSLs, such as visual GUI editors and a code generator. Visual editors support the proposed DSLs and allow application administrators to design operational policies. The code generator of iNetLab transforms operational policies defined in the proposed DSLs into program code. Currently, it generates Java code that runs on a simulator to test autonomic network applications. This code generation enables rapid configuration and implementation of operational policies of autonomic network applications, thereby alleviating the issue of information overloading. In addition, by customizing the default model-to-code transformation rule, operational policies can be transformed to other platforms (other simulators, testbeds, and real networks) without making any changes on those policies.

The third component in iNetLab is a set of performance estimators. The proposed performance estimators leverage the performance history of each network application (i.e., pairs of an operational policy and a performance result obtained in the past) and approximate the application's performance without actually deploying and running it on simulators, testbeds, or real networks. The proposed performance estimators address the issue of a lack of guidance methods to tune operational policies; they are designed to simplify the trial-and-error process for evaluating and tuning operational policies and aid application administrators to obtain reasonable operational policies that allow applications to satisfy desirable performance requirements.

This chapter is structured as follows. Section 3 presents an application architecture for autonomic networking, called BEYOND, which iNetLab is currently designed for. Section 4 overviews the architecture of iNetLab. Section 5 describes DSLs and their supporting facilities. Section 6 describes and evaluates the iNetLab performance estimators. Sections 7 and 8 conclude with some discussion on related work.

3 BEYOND: An Application Architecture for Biologically Inspired Autonomic Networking

iNetLab is currently intended to support an architecture for autonomic network applications, called BEYOND¹ [17]. This section briefly overviews BEYOND to better explain iNetLab in Sect. 5. See [17] for full discussion on BEYOND.

¹ Biologically Enhanced sYstem architecture beyond Ordinary Network Designs.

BEYOND is designed to address two challenges in autonomic network applications: *autonomy* and *adaptability*. Inspired by an observation that various biological systems have developed the mechanisms to overcome these challenges, BEYOND applies key biological principles and mechanisms to design network applications.

3.1 Agents

In BEYOND, each network application is designed as a decentralized group of software agents. This is analogous to a bee colony (application) consisting of multiple bees (agents). Each agent provides a certain functional service in an application, and implements biologically inspired behaviors. Each agent also possesses its own *behavior policy*, which determines which behavior to be invoked under a given set of environment conditions. A behavior policy in BEYOND is equivalent to an operational policy in an autonomic application. Using its behavior policy, each agent invokes its behaviors autonomously; without any intervention from/to other agents and human users. Example agent behaviors are listed below.

- **Energy exchange and storage:** Biological entities strive to seek and consume food for living. Similarly, in BEYOND, agents store and expend *energy* for living. Each agent gains energy in exchange for performing its functional service to other agents or human users, and expends energy to use the resources available at the local host (e.g., memory space and CPU cycles).
- **Replication:** Agents may make their copies in response to high energy level, which indicates high demand for the agents. A replicated agent is placed on the host that its parent agent resides on, and it inherits the parent's behavior policy. Mutation may occur on the inherited behavior policy.
- **Reproduction:** Agents may reproduce child agents with other agents (mating partners). A child agent is placed on the host that its parents reside on, and it inherits behavior policies from both parents through crossover. Mutation may occur on the behavior policy of a child agent.
- **Migration:** Agents may move from one network host to another.
- **Death:** Agents die due to energy starvation. If an agent cannot balance its energy expenditure with its energy gain, the agent cannot pay for the resources it needs; thus, it dies from lack of energy.

3.2 iNet: Agent Adaptation Mechanism in BEYOND

iNet is a key component in BEYOND, which allows each agent to adaptively perform its behaviors against dynamic environment conditions in the network, such as network traffic and resource availability. iNet is designed after the mechanisms behind how the immune system detects antigens (e.g., viruses), how it specifically produces antibodies to eliminate them, and how it evolves antibodies to react to a massive number of antigens. iNet models a set of environment conditions as an

antigen and an agent behavior as an antibody. Each agent contains its own immune system, and a configuration of the agent's antibodies defines its behavior policy. iNet allows each agent to autonomously sense its surrounding environment conditions (i.e., antigen) for evaluating whether it adapts well to the sensed conditions, and if it does not, adaptively invoke a behavior (i.e., antibody) suitable for the conditions. For example, agents may invoke the replication behavior at the network hosts that accept a large number of user requests for their services. This leads to the adaptation of agent availability; agents can improve their throughput. Also, agents may invoke the migration behavior to move toward the network hosts that receive a large number of user requests for their services. This results in the adaptation of agent locations; agents can improve their response time to user requests.

3.2.1 Natural Immune System

The natural immune system adaptively regulates the body against dynamic environmental changes such as antigen invasions. Through a number of interactions among various white blood cells (e.g., macrophages and lymphocytes such as T-cells and B-cells) and molecules (e.g., antibodies), the immune system evokes two responses to antigens: *T-cell activation* and *B-cell activation* responses.

In the T-cell activation response, the immune system performs self/non-self discrimination. This response is initiated by macrophages. Macrophages move around the body to ingest antigens and present them to T-cells. T-cells are produced in thymus though the negative selection. In this selection process, thymus removes T-cells that strongly react to the body's own (self) cells. The remaining T-cells are used as detectors to identify foreign (non-self) cells. When a T-cell detects a non-self cell presented by a macrophage, the T-cell secretes chemical signals to induce the B-cell activation response.

In the B-cell activation response, B-cells are activated by T-cells. Some of the activated B-cells strongly react to an antigen, and they produce antibodies that specifically kill the antigen. Antibodies form a network and communicate with each other [11]. This antibody network is formed with stimulation and suppression relationships among antibodies. With these relationships, antibodies dynamically change their populations and network structure. For example, the population of a specific type of antibodies rapidly increases when they detect an antigen, and after eliminating the antigen, the population decreases again.

The immune system maintains approximately 10^9 antibodies. B-cells can increase this repertoire further by mutating and recombining immune gene segments so that antibodies can detect and bind a massive number of antigens [3].

3.2.2 iNet Artificial Immune System

The iNet artificial immune system consists of the environment evaluation (EE) facility and behavior selection (BS) facility, which implement the T-cell and B-cell activation responses, respectively (Fig. 1). The EE facility allows an agent to continuously sense a set of current environment conditions as an antigen and classify

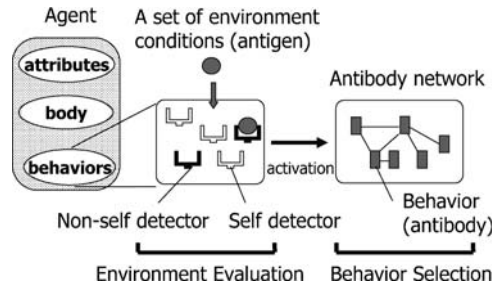


Fig. 1 The architecture of iNet

the antigen to self or non-self. A self antigen indicates that the agent adapts to the current environment conditions well, and a non-self antigen indicates it does not. When the EE facility detects a non-self antigen, it activates the BS facility. The BS facility allows an agent to choose a behavior as an antibody that specifically matches the detected non-self antigen.

The EE facility performs two steps: initialization and self/non-self classification. The initialization step produces detectors, as T-cells, which identify self and non-self antigens. Each antigen is represented as a feature vector (X), which consists of a set of environment conditions, or features, (F_i) and a class value (C):

$$X = (F_1, F_2, \dots, F_n, C) \tag{1}$$

C indicates whether a given antigen (i.e., a set of environment conditions) is self (0) or nonself (1). If an agent senses resource utilization and workload (the number of user requests) on the local host, an antigen is represented as

$$X_{current} = ((Low : ResourceUtilization, Light : Workload), 0) \tag{2}$$

The initialization of the EE facility is designed after the negative selection in the immune system (Fig. 2). As the immune system randomly generates T-cells, the EE facility generates detectors (feature vectors) randomly. Then, the EE facility separates the generated detectors into self detectors, which closely match self antigens, and non-self detectors, which do not. This separation is performed by measuring vector similarity between randomly generated feature vectors (R) and self antigens (S) that human administrators supply. After the vector matching, both self and non-self detectors are stored in the detector table (Fig. 2).²

In self/non-self classification, the EE facility classifies a given antigen to self or non-self. This is performed with a decision tree built from the detectors in the detector table. Figure 3 shows an example decision tree. Each node in the tree specifies which feature (environment condition) is considered. Based on the feature values

² The immune system removes non-self detectors through negative selection. However, in iNet, both self and non-self detectors are used to perform self/non-self classification.

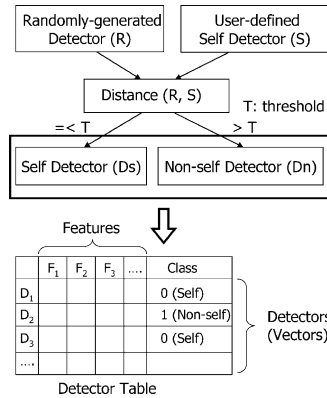


Fig. 2 Initialization of the EE facility

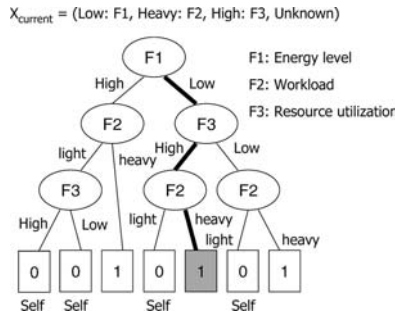


Fig. 3 An example decision tree

in a given antigen, the EE facility travels through tree branches. If the EE facility classifies the antigen to non-self, it activates the BS facility.

The BS facility selects an antibody (i.e., agent’s behavior) suitable for a detected non-self antigen (i.e., a set of environment conditions). Each antibody consists of three parts: a *precondition* under which it is selected, *behavior ID* and *relationships* to other antibodies. Antibodies are linked with each other using stimulation and suppression relationships. Each antibody has its own concentration value, which represents its population. The BS facility identifies a set of antibodies suitable for a given non-self antigen, prioritizes them based on their concentration values, and selects the most suitable one. When prioritizing antibodies, stimulation relationships among them contribute to increase their concentration values, and suppression relationships contribute to decrease them. Each relationship has an affinity value, which indicates the degree of stimulation or suppression.

Figure 4 shows an example network of antibodies. It contains four antibodies, which represent the migration, replication and death behaviors. Antibody 1 represents the migration behavior invoked when the distance to users is far from an agent. Antibody 1 suppresses Antibody 3 and stimulates Antibody 4. Now, suppose that a

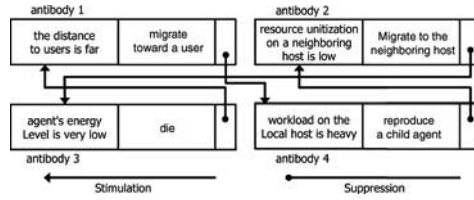


Fig. 4 An example antibody network

(non-self) antigen indicates (1) the distance to users is far, (2) workload is heavy on the local host, and (3) resource utilization is low on a neighboring platform. This antigen stimulates Antibodies 1, 2, and 4 simultaneously. Their populations increase, and Antibody 2’s concentration value becomes highest because Antibody 2 suppresses Antibody 4, which in turn suppresses Antibody 1. As a result, it is likely that the BS facility selects Antibody 2.

4 The Architecture of iNetLab

This section overviews the architecture of iNetLab. It provides a development, configuration, and performance engineering environment for autonomic network applications built with iNet. Figure 5 shows an architectural overview of iNetLab. It consists of six components: four application configuration facilities (Sect. 5), application code generator (Sect. 5), and performance estimators (Sect. 6). The iNetLab configuration facilities aid defining and configuring iNet-based applications with visual/textual DSLs. They include the environment configuration facility (Sect. 5.1),

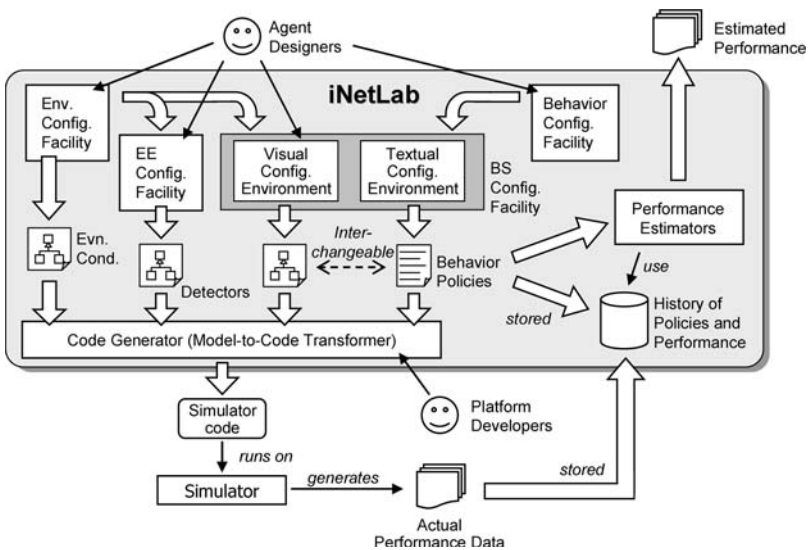


Fig. 5 The architecture of iNetLab

the agent behavior configuration facility (Sect. 5.2), the EE configuration facility (Sect. 5.3), and the BS configuration facility (Sect. 5.4).

The environment configuration facility allows agent designers (i.e. application designers) to visually configure the environment conditions used in their agents. The behavior configuration facility allows them to visually configure agent behaviors. The EE configuration facility allows for configuring a set of self detectors (S in Fig. 2) used in the EE facility. The BS configuration facility allows agent designers to visually or textually configure the behavior policies of their agents.

Once environment conditions, agent behaviors, detectors, and behavior policies are configured with the iNetLab configuration facilities, the iNetLab code generator transforms them to compilable source code with a DSL-to-code transformation rule. Transformation rules are implemented by platform developers, who know the details of platform technologies to run applications (e.g., programming languages, operating systems, middleware, and simulators). Through changing one transformation rule to another, the iNetLab code generator can generate different source code that are compatible with different deployment environments such as simulators and real networks. This way, an agent designer can define a single set of application configurations and reuse it for different platform technologies. Currently, iNetLab supports Java code generation for a simulator of BEYOND.

After generating an application code and running it on a simulator, iNetLab collects the application's performance result. Then, it stores a pair of the application's behavior policy and performance result in a repository as history data. The iNetLab performance estimators use the history data to approximate application performance with new behavior policies in the future.

5 DSLs, Configuration Facilities and Code Generator in iNetLab

This section describes four configuration facilities and code generator in iNetLab.

5.1 iNetLab Environment Configuration Facility

The iNetLab environment configuration facility allows agent designers to visually model environment conditions with its DSL. Figure 6 shows an example environment condition model. As this figure illustrates, each rectangle represents an environment condition and contains multiple rounded rectangles that represent its value categories. For example, in Fig. 6, the `LocalWorkload` environment condition defines two value categories: `HEAVY` (higher than 200) and `LIGHT` (lower than or equal to 200).

It is hidden from agent designers how and when to obtain values of environment conditions. Platform developers are expected to implement this concern in a skeleton source code generated by the iNetLab code generator (Fig. 5). For example, Listing 1 shows a fragment of Java code generated from the `LocalWorkload` environment condition in Fig. 6. The class `EnvironmentCondition` is the base

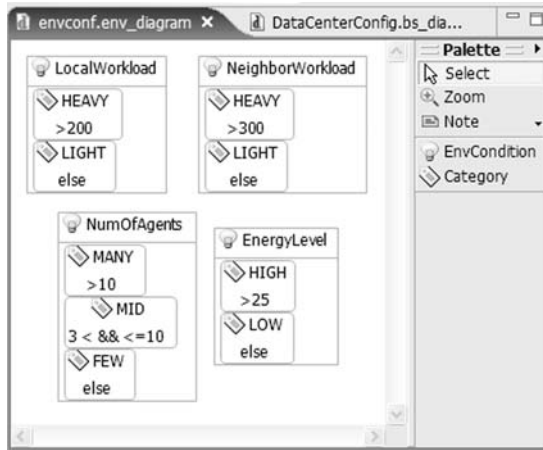


Fig. 6 An example model defined with the iNetLab environment configuration facility

class to define environment conditions used in a BEYOND simulator; it provides a means to obtain values of environment conditions by accessing the states of the simulator. Platform developers implement the `getRepValue()` method with the APIs in `EnvironmentCondition`. For example, the APIs are used to return the current CPU utilization and request rate from users.

Figure 7 shows the metamodel of environment condition models. Any environment condition model (e.g., Fig. 6) is defined as an instance of this metamodel. Other configuration facilities and code generator access environment condition models via this metamodel.

The metamodel has three metaclasses, `EnvironmentConditionDefs`, `EnvironmentConditionDef`, and `CategoryDef`. `EnvironmentConditionDefs` represents a set of environment condition definitions. This metaclass does not have its graphical notation because its instance corresponds to an environmental configuration model itself. An instance of the `EnvironmentConditionDef` metaclass represents the definition of an environment condition (e.g., `LocalWorkload` in Fig. 6), and its name attribute indicates the name of an environment condition (e.g., "LocalWorkload" in Fig. 6). An instance of

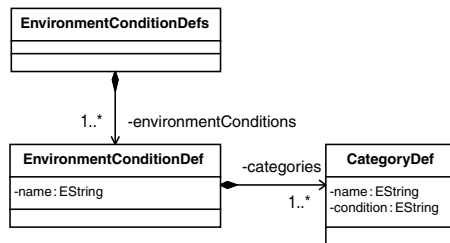


Fig. 7 Metamodel for environment condition models

EnvironmentConditionDef metaclass can contains an arbitrary number of instances of the CategoryDef metaclass, which defines a value category of an environment condition. The name attribute indicates a category's name, and the expression attribute indicates a value range of an environment condition. For example, in Fig. 6, LocalWorkload has a value category, called Heavy, whose range is ">200. "

Listing 1 An Example Generated Code for the LocalWorkload Environment Condition

```

1 public class LocalWorkload
2     extends edu.umb.inet.sim.EnvironmentCondition
3     implements EnvironmentCondition {
4
5     enum Category{ HEAVY, LIGHT };
6
7     public Category evaluate(){
8         double repValue = getRepValue();
9         if( repValue > 200 ){ return Category.HEAVY;}
10        return Category.LIGHT;
11    }
12
13    private double getRepValue(){
14        // TODO: platform developers add code here
15    }
16 }

```

The metamodel for environment condition models is defined in Eclipse Modeling Framework (EMF) [9], and the environment configuration facility is implemented on Eclipse Graphical Modeling Framework (GMF) [8].

The iNetLab code generator transforms an environment condition model to Java source code with a DSL-to-code transformation rule (Fig. 5). The transformation rule is defined as a template that maps the metamodel elements of environment condition models and the program elements of Java source code. Each rule is executed with openArchitectureware (oAW) [19], a model-to-code transformation engine. Listing 2 shows a fragment of the default transformation rule used for environment condition models. This rule generates Java source code used in a BEYOND simulator. Platform developers can define their own transformation rules that generate source code for other deployment environments (Fig. 5).

Listing 2 A Fragment of the Default Transformation Rule for Environment Condition Models

```

1 <<DEFINE ExpandEnvCondition FOR EnvironmentConditionDef>>
2 <<FILE name + ".java">>
3 public class <<name>>
4     extends edu.umb.inet.sim.EnvironmentCondition
5     implements EnvironmentCondition {
6
7     enum Category{
8         // Apply RetrieveCategoryName to each element
9         // in EnvironmentConditionDef.categories
10        <<EXPAND RetrieveCategoryName FOREACH categories>> };
11
12    public Category evaluate(){
13        double repValue = getRepValue();
14        <<EXPAND ExpandCondition FOREACH categories>>
15    }
16    ...
17 }
18 <<ENDFILE>>
19 <<ENDDDEFINE>>

```

```

1 // Retrieve the 'name' attribute of CategoryDef
2 <<DEFINE RetrieveCategoryName FOR CategoryDef>>
3   <<name>>,
4 <<ENDDDEFINE>>
5
6 // Transform a category to an if statement according to its condition
7 <<DEFINE ExpandCondition FOR CategoryDef>>
8   <<IF condition == "else">>
9     return <<name>>;
10  <<ELSE>>
11    if(<<condition>>) return <<name>>;
12  <<ENDIF>>
13 <<ENDDDEFINE>>

```

The keyword `DEFINE` defines a transformation rule for a certain metaclass. In Line 1 of Listing 2, a transformation rule, named `ExpandEnvCondition`, is defined for the metaclass `EnvironmentConditionDef`. Each instance of the metaclass is transformed to a Java class whose name is same as the instance's attribute name (`<<name>>` is replaced with the instance's attribute name.) In Line 7, a Java enumeration type (`Category`) is defined. Its elements are defined by calling the `RetrieveCategoryName` rule (Lines 22–24), which retrieves name of each instance of `CategoryDef`. In Line 12, the `evaluate()` method is defined, and completed by calling the `ExpandCondition` rule on each instance of `CategoryDef`.

This transformation rule generates Java source code shown in Listing 1 when it is applied to an environment condition model shown in Fig. 6.

5.2 *iNetLab Behavior Configuration Facility*

The *iNetLab* behavior configuration facility allows agent designers to visually model agent behaviors with its DSL. Figure 8 shows an example behavior configuration model. As this figure illustrates, each rectangle represents an agent behavior and contains multiple rounded rectangles that represent its parameters. The name of a parameter is shown at the top of a rounded rectangle, and the parameter's type is shown below. For example, in Fig. 8, the `Reproduction` behavior has three parameters: `mutationRate`, `partnerSelectionPolicy`, and `crossoverPolicy`. A parameter can be typed with an enumeration. In



Fig. 8 An example model defined with the *iNetLab* behavior configuration facility

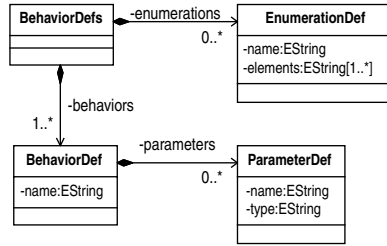


Fig. 9 Metamodel for behavior configuration models

Fig. 8, two enumeration types are defined: PartnerSelectionPolicy and CrossoverPolicy.

Figure 9 shows the metamodel for behavior configuration models. Any behavior configuration model (e.g., Fig. 8) is defined as an instance of this metamodel. Other configuration facilities and code generator access behavior configuration models via this metamodel.

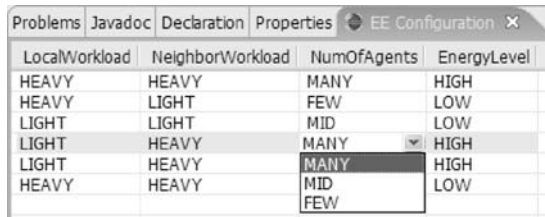
The metamodel consists of four metaclasses: BehaviorDefs, BehaviorDef, ParameterDef, and EnumerationDef. BehaviorDefs represents a set of agent behavior definitions. This metaclass does not have its graphical notation because its instance corresponds to a behavior configuration model itself. An instance of the BehaviorDef metaclass represents the definition of an agent behavior (e.g., Reproduction in Fig. 8), and its name attribute indicates the name of an agent behavior (e.g., "Reproduction" in Fig. 8). An instance of the BehaviorDef can contain an arbitrary number of instances of the ParameterDef metaclasses, which defines the parameters of an agent behavior. The name and type of ParameterDef represent a parameter's name and type.

Similar to the environment configuration facility, the behavior configuration facility is implemented on Eclipse GMF. The metamodel for behavior configuration models is defined in EMF.

5.3 iNetLab EE Configuration Facility

The iNetLab EE configuration facility allows agent designers to define a set of self detectors (*S* in Fig. 2) used in the EE facility. Figure 10 shows a screenshot of this facility, and depicts six detectors in a table. Each row in the table represents a detector, and each column represents an environment condition defined in the environment configuration facility (Sect. 5.1).

In the EE configuration facility, agent designers configure detectors by selecting one of the categories for each environment condition. For example, in Fig. 10, the NumOfAgents environment condition has three categories, MANY, MID, and FEW, which are defined in the environment configuration facility (Fig. 6). An agent designer chooses one of the three categories to generate detectors. As described in Sect. 3.2.2, the generated detectors are used to perform the negative selection process in iNet.



LocalWorkload	NeighborWorkload	NumOfAgents	EnergyLevel
HEAVY	HEAVY	MANY	HIGH
HEAVY	LIGHT	FEW	LOW
LIGHT	LIGHT	MID	LOW
LIGHT	HEAVY	MANY	HIGH
LIGHT	HEAVY	MANY	HIGH
HEAVY	HEAVY	MID	LOW
		FEW	

Fig. 10 Example detectors defined with the iNetLab EE configuration facility

5.4 iNetLab BS Configuration Facility

The BS configuration facility allows agent designers to visually or textually configure the behavior policies of their agents. Figure 11 shows a visual behavior policy (antibody network) model. As this figure illustrates, each rectangle represents an antibody and consists of three compartments: (1) the name and the initial concentration of an antibody, (2) an environment condition under that an antibody reacts to, and (3) an agent behavior and its parameters. For example, in Fig. 11, AntibodyA represents the reproduction behavior, and its initial concentration value is 5. The behavior is invoked when LocalWorkload is light. A stimulation/suppression relationship between antibodies is visualized as a solid arrow between rectangles. Each arrow shows a value that represents the affinity value of a corresponding stim-

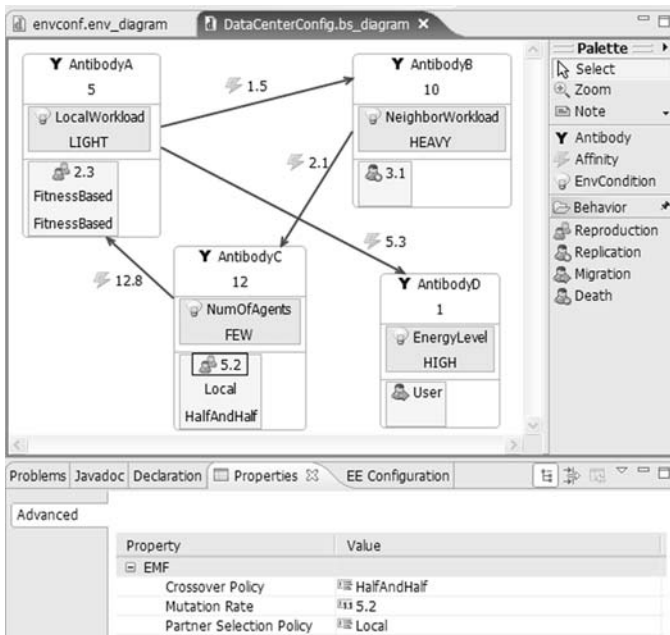


Fig. 11 A visual behavior policy model defined with the iNetLab BS configuration facility

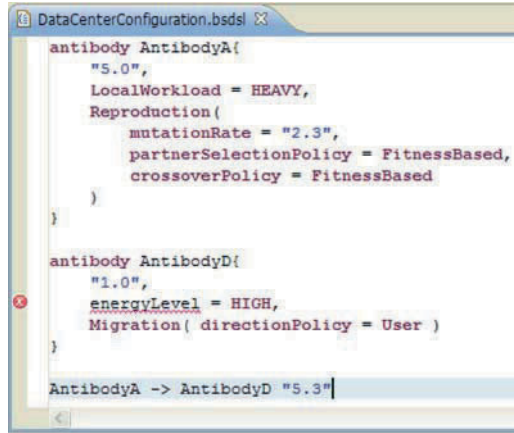


Fig. 12 A textual behavior policy model defined with the iNetLab BS configuration facility

ulation/suppression relationship. In Fig. 11, AntibodyA stimulates AntibodyB with the affinity value of 1.5.

Figure 12 shows a textual behavior policy configuration. Each antibody is defined with the built-in keyword `antibody`. Figures 11 and 12 show the semantically same behavior policy. As in Fig. 12, the BS configuration facility shows built-in keywords in a boldface, automatically examine the syntax of a behavior policy configuration, and reports syntax errors while agent designers configure antibodies. In Fig. 12, a syntax error is reported with a cross mark. (The keyword `energyLevel` is wrong; `EnergyLevel` should be used because of the environment condition model defined in Fig. 6.)

Listing 3 is a fragment of Java source code to which the iNet code generator transforms from the behavior policy configuration in Fig. 11 or 12. Different forms (visual and textual) of a behavior policy configuration are transformed to the same source code.

Listing 3 An Example Generated Code for Configuring an Antibody Network

```
1 void setupAntibodiesOfINet(){
2     Antibody antibodyA =
3     new Antibody( "AntibodyA", 5, LocalWorkload.LIGHT,
4     new Reproduction(
5     2.3, CROSSOVER.FITNESSBASED, PARTNER.FITNESSBASED ) );
6
7     Antibody antibodyD =
8     new Antibody( "AntibodyD", 1, EnergyLevel.HIGH,
9     new Migration( DirectionPolicy.USER ) );
10
11     AntibodyNetwork inet = getAntibodyNetwork();
12     inet.add( antibodyA );
13     inet.add( antibodyD );
14     antibodyA.addAffinity( antibodyD, 5.3 );
15 }
```

The BS configuration facility allows agent designers to configure behavior policies (antibody networks) in a declarative and intuitive manner. They do not need

to know the programming details on how to implement agents in Java (e.g., how to define agents, where to implement a behavior policy in agent code, and which iNet APIs to use for implementing antibodies.) These details are hidden from agent designers by the BS configuration facility and code generator; they can focus on the design of behavior policies.

Figure 13 shows the metamodel for behavior policy models. It consists of five generic metaclasses, *AntibodyNetwork*, *Antibody*, *Affinity*, *Behavior*, and *EnvironmentCondition*, and four metaclasses for agent behaviors, *Reproduction*, *Replication*, *Migration*, and *Death*.

AntibodyNetwork represents an antibody network. For example, a behavior policy model in Fig. 11 is an instance of *AntibodyNetwork*. *Antibody* represents an antibody, and its *name* and *initialConcentration* attributes indicate an antibody’s name and initial concentration value, respectively. *Affinity* represents an affinity between *Antibodys*; the direction and degree of a stimulation/suppression relationship. *Behavior* is the base metaclass for all agent behaviors. By referencing *Environment* and *Category*, *EnvironmentCondition* represents an environmental condition under which an antibody is invoked.

The visual and textual BS configuration environments are implemented with Eclipse GMF and oAW, respectively. The metamodel for behavior policy configurations is defined in EMF.

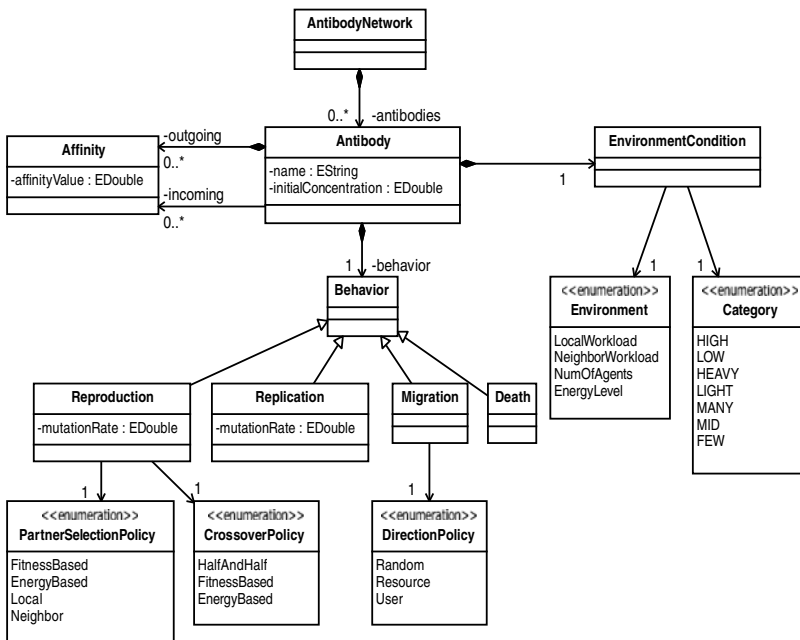


Fig. 13 The metamodel for behavior policy models

5.5 Metamodel Customization

As described in earlier sections, the environment configuration metamodel (Fig. 7), agent behavior configuration metamodel (Fig. 9), and behavior policy configuration metamodel (Fig. 13) are built upon EMF, which serves as the meta-metamodel (Fig. 14). Similarly, environment configuration models (e.g., Fig. 6), agent behavior configuration models (e.g., Fig. 8) and behavior policy models (e.g., Figs. 11 and 12) are built upon their corresponding metamodels (Fig. 14).

In iNetLab, the behavior policy configuration metamodel (Fig. 13) is intended to be extensible for various types of applications that consider different environment conditions and use different agent behaviors. In other words, the metamodel varies depending on what are defined in environmental configuration models and agent behavior configuration models. In order to address this issue and make the metamodel extensible, iNetLab customizes the metamodel based on given environmental configuration models and agent behavior configuration models (Fig. 14).

The first step of metamodel customization is to import an environment configuration model from the environment configuration facility, extract environment conditions defined in the model, and introduce the environment conditions to the behavior policy configuration metamodel. For example, when an environment configuration model in Fig. 6 is imported, four environment conditions (e.g., LocalWorkload) and seven category values (HEAVY and LIGHT) are extracted. Then, the behavior policy configuration metamodel is customized by generating two enumeration types (Environment and Category) and defining four environment conditions and seven category values in the enumeration types. See Fig. 13 for the generated enumeration types.

The second step of metamodel customization is to import an agent behavior model from the behavior configuration facility, extract behaviors defined in the model, and introduce the behaviors to the behavior policy configuration metamodel. For example, when an agent behavior model in Fig. 8 is imported, the Reproduction behavior and its associated parameters (i.e., mutationRate,

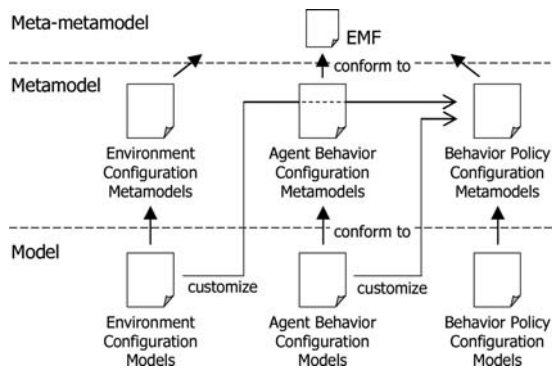


Fig. 14 Metamodel customization in iNetLab

partnerSelectionPolicy, and crossoverPolicy) are extracted. Then, the behavior policy configuration metamodel is customized by generating a sub metaclass of the Behavior metaclass and defining three parameters in the generated metaclass. In Fig. 13, four behaviors are generated and defined, including Reproduction.

Agent designers can anytime change the environment conditions and behaviors that their agents (applications) use by re-defining environment configuration models and behavior configuration models and re-performs metamodel customization. This way, they can customize the behavior policy metamodel without knowing meta-modeling details and maintain the metamodel extensible yet consistent with other models.

6 Performance Estimators in iNetLab

The iNetLab performance estimators implement two different estimation methods using graph similarity (Sect. 6.1) and eigenvector centrality (Sect. 6.2). As illustrated in Fig. 5, iNetLab records, as history data, pairs of behavior policies used in an application (i.e., agents) and the application's performance results. Once an agent designer configures a new behavior policy for an application (agents), an iNetLab performance estimator retrieves the behavior policies, from the history data, which are similar to the one under development. Then, it approximates the application's performance under an assumption that similar behavior policies yield similar performance results. With the iNetLab performance estimators, agent designers can re-define and tune their behavior policies without running applications (agents) repeatedly for a long time. This can alleviate trial-and-error burdens from agent designers and improve their productivity in tuning behavior policies.

6.1 Performance Estimation with Graph Similarity

The first performance estimation method in iNetLab is inspired by a bioinformatics technique to understand and infer the function of a network of interacting proteins. In biology, protein interaction networks have been extensively investigated

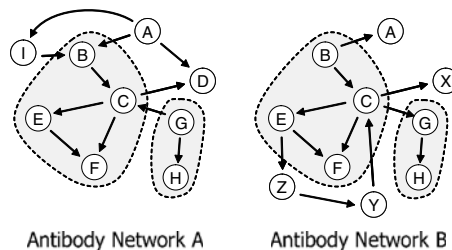


Fig. 15 Common subnetworks in two behavior policies (antibody networks)

to explore how interacting proteins reveal an emergent function such as creating a membrane and signaling impulses between cells. It is now known that similar protein interaction networks have similar functions even if several proteins and interaction patterns are altered. Therefore, by measuring structural similarity with other networks, it is possible to infer the function of a given protein interaction network. In bioinformatics, several functional approximation methods have been proposed.

Since antibodies are proteins and an antibody network has an emergent function (i.e., immune response) through stimulation/suppression interactions, a performance estimator in iNetLab is designed based on an approximation method for protein interaction networks [22]. In this performance estimator, called the iNet graph-based performance estimator, the function of an antibody network corresponds to the performance of an application that uses the antibody network as its behavior policy.

The proposed graph-based estimator approximates an application’s performance by measuring the structural similarity (or graph similarity) between the application’s behavior policy and other behavior policies. It measures the similarity, S , and dissimilarity, D , between two different behavior policies, and obtains $(S - D)$ as the overall similarity S_{overall} . S is measured by finding the common subnetworks contained (or shared) in given two behavior policies. D is measured by finding the subnetworks that are not shared in two behavior policies. A common subnetwork is a network that consists of the same types of antibodies (i.e., antibodies that represent the same behavior, have the same environment conditions and have the same directed structure/graph of stimulation/suppression relationships). For example, in Fig. 15, two behavior policies have two common subnetworks: a subnetwork consisting of B, C, E, and F, and another subnetwork consisting of G and H. In the Antibody Network A, a subnetwork consisting of A, D, and I is not shared with the Antibody Network B. In the Antibody Network B, a subnetwork consisting of A, X, Y, and Z is not shared with the Antibody Network B.

The similarity, S , between two behavior policies is calculated with Eq. (3) where $anetA$ and $anetB$ represent behavior policies. When two behavior policies are identical, the similarity between them is equal to the number of antibodies and stimulation/suppression relationships.

$$\begin{aligned}
 S(anetA, anetB) = & \sum_{ab \in AB} \max(0, 1 - |I_{anetA}(ab) - I_{anetB}(ab)|) \\
 & + \sum_{r \in R} \max(0, 1 - |A_{anetA}(r) - A_{anetB}(r)|) \quad (3)
 \end{aligned}$$

where

AB = A set of antibodies in $anetA \cap anetB$

R = A set of stimulation/suppression relationships in $anetA \cap anetB$

$I_{anet}(ab)$ = The initial concentration of an antibody ab in $anet$.

$A_{anet}(r)$ = Affinity value associated with a relationship r in $anet$.

The dissimilarity, D , between two behavior policies is calculated with Eq. (4). When two behavior policies are identical, the dissimilarity between them becomes 0. When the antibodies and relationships that are not shared in two behavior policies have larger initial concentration values and affinity values, the dissimilarity between them becomes larger.

$$D(anetA, anetB) = \sum_{ab' \in AB'} I(ab') + \sum_{r' \in R'} A(r') \tag{4}$$

where

$AB' = A$ set of antibodies in $anetA \Delta anetB$

$R' = A$ set of stimulation/suppression relationships in $anetA \Delta anetB$.

6.2 Performance Estimation with Eigenvector Centrality

The second performance estimation method in iNetLab is designed to predict the concentrations of antibodies in each antibody network and obtain the similarity of behavior policies (i.e., antibody networks) by comparing the predicted antibody concentrations in given two antibody networks. This estimation method leverages eigenvector centrality [4, 5] to predict the concentrations of antibodies.

The proposed eigenvector-based method forms an $N \times N$ adjacent matrix that represents individual stimulation/suppression relationships between antibodies, when N antibodies exist in an antibody network. Each value in the matrix represents an affinity value associated with a stimulation/suppression relationship. For example, in Fig. 16, the antibody C stimulates the antibody D with the affinity value of 1.2. The antibody A's initial concentration is 0.0, and the antibody B's is 1.0.

The proposed method obtains multiple N -dimensional eigenvectors from a given adjacent matrix and choose the eigenvector, called *centrality vector*, which generates the maximum eigenvalue. In general, a centrality vector provides N *centrality*

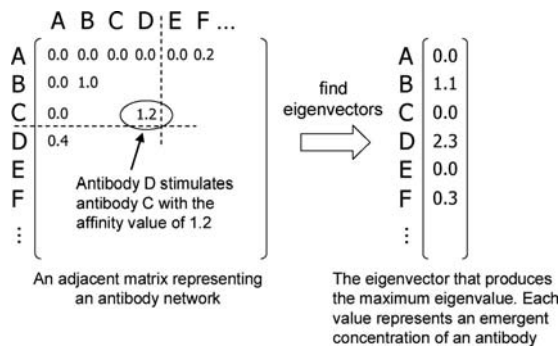


Fig. 16 Adjacent matrix and eigenvector of a behavior policy (antibody network)

scores, each of which represents a certain importance level [5]. In the context of iNetLab, these centrality scores indicate the emergent concentrations of antibodies through stimulation and suppression interactions among antibodies. For example, in Fig. 16, the antibody B’s concentration is predicted as 1.1.

Since an antibody’s concentration impacts the probability that it is selected (Sect. 3.2.2), a centrality vector indicates how an application (agents) invokes behaviors with a given behavior policy and, in turn, estimates the application’s performance.

The proposed eigenvector-based performance estimation method examines the similarity of behavior policies by comparing their centrality vectors based on the cosine similarity.

6.3 Evaluation of Performance Estimators

This section presents a series of simulation results to evaluate the accuracy of iNetLab performance estimators.

6.3.1 Simulation Configurations

The simulations were carried out on a BEYOND simulator. Figure 17 shows a simulated server farm consisting of network hosts connected in a 3 × 3 grid topology. Each agent is simulated to provide a web service that receives a service request message and returns an HTML file. Service requests travel from users to agents via user access point. This simulation study assumes that a single (virtual) user runs on the access point and sends request messages to agents.

Figure 18 shows how the user changes service request rate over time. This is designed based on a workload trace of the www.ibm.com site in February 2001 [6]. The workload falls down to 3,000 requests per minute in early morning, and peaks 7,500 requests per minute in the afternoon. At the beginning of simulations, one agent is deployed on a randomly selected host. This simulation generates a workload trace that is designed based on a daily request rate for the www.ibm.com site in February, 2001 [6].

When a simulation is completed, iNetLab records performance results with the following five metrics:

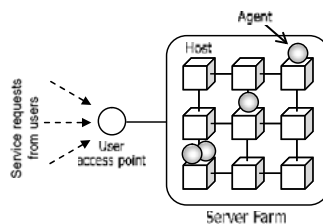


Fig. 17 Simulated server farm

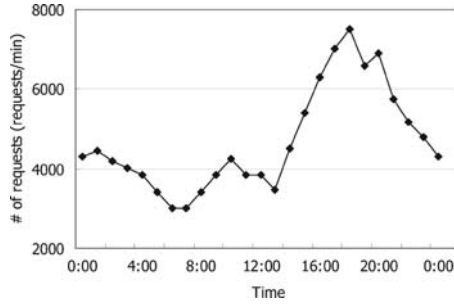


Fig. 18 Simulated workload

- Throughput: $\frac{\text{The total number of messages that agents process}}{\text{The total number of messages that the user transmitted}}$
- Resource efficiency: $\frac{\text{The number of messages that agents process}}{\text{The amount of resources that agents consume}}$
- Distance: The average hop count from agents to the user
- Latency: The average of latency
- Latency Jitter: Variance of latency

This simulation study considers 15 environment conditions and 5 agent behaviors. Since a behavior policy contains an arbitrary combination of 75 antibodies, the total number of possible combinations is $\sum_{m=1}^{75} C_m = 2^{75} \cong 3.7 \times 10^{22}$. The number of combination becomes far larger than 3.7×10^{22} if different affinity values are considered for relationships between antibodies.

As this example illustrates, the number of possible behavior policies is astronomical numbers even if a few environment conditions and agent behaviors are considered. It is unpractical and nearly impossible to tune application performance by testing all of them one by one in simulations. Therefore, the next section evaluates how accurately iNetLab performance estimators approximate application performance with a relatively small amount of history data (i.e., a relatively small number of actual simulation runs).

6.3.2 Simulation Results

In this simulation study, iNetLab randomly generates 1,000 behavior policies, run an application with them on a BEYOND simulator and records performance results as history data. Given a behavior policy BP_g , each iNetLab performance estimator finds the most similar behavior policy BP_s from history data. The accuracy of each performance estimator is measured with the performance resulting from BP_g , called PF_g , is very close to the performance resulting from BP_s , called PF_s . The closeness of PF_s and PF_g is calculated with Eq. (5).

$$C(PF_s, PF_g) = N - \sum_{i=1}^N \frac{|(metric_i \text{ in } PF_s) - (metric_i \text{ in } PF_g)|}{\text{the max value in } metric_i}$$

where

$$N = \text{The total number of metrics} \tag{5}$$

In order to measure the estimation accuracy of two different performance estimators, each performance estimator (1) takes a randomly generated behavior policy BP_g , (2) runs an application with BP_g in a simulator and obtains a set of performance results PF_g , (3) sorts the 1,000 behavior policies in history data in order of the closeness of their performance results to PF_g , (4) finds BP_s , which is most similar to BP_g , with a certain estimation method and obtains FP_s as estimated performance, and (5) determines the rank of PF_s in the list obtained at (3). When the rank of PF_s (estimated performance) is 1, the accuracy of a performance estimator is highest.

Figures 19 and 20 show the accuracy of graph-based and eigenvector-based estimators, respectively. For each figure, each estimator runs performance approximation 1,000 times and counts the rank of PF_s as frequency. Each figure shows this

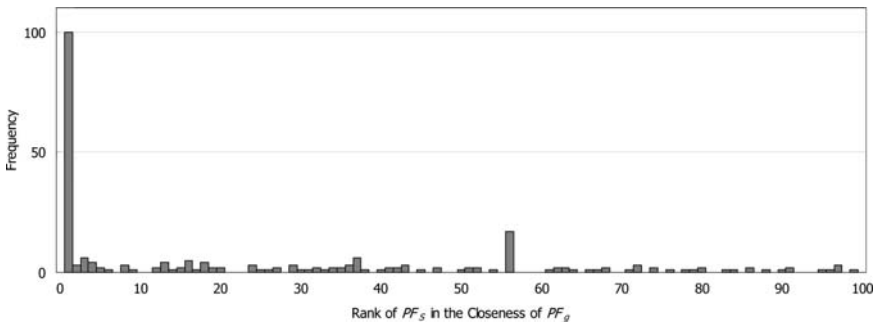


Fig. 19 Estimation accuracy of graph-based performance estimation

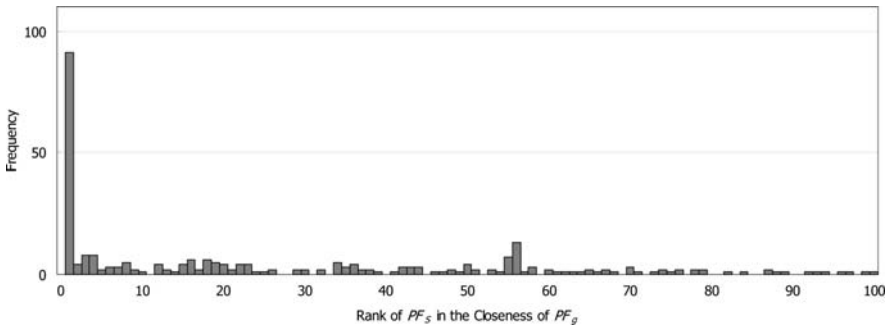


Fig. 20 Estimation accuracy of eigenvector-based performance estimation

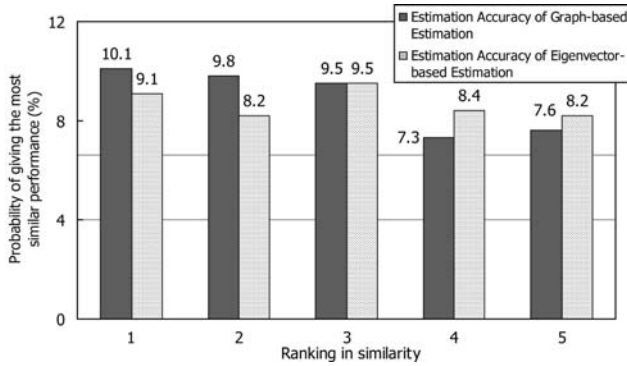


Fig. 21 Estimation accuracy of of graph-based and eigenvector-based performance estimation

frequency in its Y -axis.³ As these two figures show, both performance estimators can accurately identify the BP_s that yields the closest performance to the actual performance (i.e., the first rank PF_s).

Figure 21 shows the estimation accuracy of graph-based estimation. It shows the probabilities that the five most similar behavior policies in history data yield the closest performance of a given behavior policy. For example, the probability that the most similar behavior policy gives the closest performance result is 10.1%. The probability that the second similar behavior policy gives the closest performance result is 9.8%. By examining the five most similar behavior policies, application developers (agent designers) can predict the application's performance with a behavior policy under development at the probability of 54.3%.

Figure 21 also shows the estimation accuracy of eigenvector-based estimation. It shows the probabilities that the five most similar behavior policies in history data yield the closest performance of a given behavior policy. For example, the probability that the most similar behavior policy gives the closest performance result is 9.1%. By examining the five most similar behavior policies, application developers (agent designers) can predict the application's performance with a behavior policy under development at the probability of 52.2%.

Figure 22 shows the number of simulation runs required to find a behavior policy that can yield 99% throughput. Without the iNetLab performance estimators, each application developer (agent designer) runs an application (agents) with a given behavior policy on a BEYOND simulator and obtains its throughput performance. If the performance does not reach 99%, the developer slightly change the behavior policy to obtain throughput performance. He/she continue this trial-and-error process until throughput performance reaches 99%. As Fig. 22 illustrates, 22

³ Although each of Figs. 19 and 20 has 1,000 different ranks in the X -axis, it shows the 1st to the 100th ranks only because frequency (Y -axis) is always less than 3 between the 101th to the 1,000th ranks.

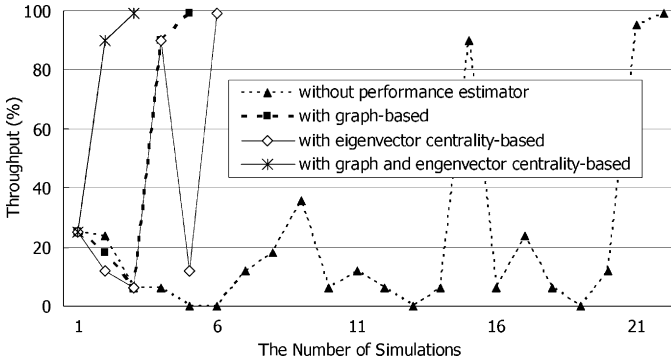


Fig. 22 The number of simulations

simulation runs are necessary to determine a behavior policy that can yield 99% throughput.

In contrast, the iNetLab performance estimators can approximate whether an application (agents) can likely yield desirable performance (i.e., 99% throughput) without running simulations. In this simulation study, an application developer do not run a simulation when an iNetLab performance estimator predicts that an application yields 10% or less throughput with a given behavior policy. As Fig. 22 shows, graph-based estimation dramatically reduces the number of necessary simulation runs from 22 to 5 in order to achieve 99% throughput. Similarly, eigenvector-based estimation reduces the number of simulations from 22 to 6. Moreover, if a developer runs a simulation only when both graph-based and eigenvector-based estimation methods predict that an application yields higher than 10% throughput, the number of necessary simulations is reduced to 3. Since these two estimation methods conduct performance estimation in different ways, estimation accuracy improves by using both at a time.

7 Related Work

This chapter describes a set of extensions to prior work [17, 24]. One extension is to investigate the iNet performance estimators, which [17, 24] do not consider. Another extension is to study metamodel customization, which [24] does not consider.

Several research efforts have investigated model-driven development techniques for autonomic computing based on general-purpose modeling languages such as UML [13, 20, 21, 23]. Their model tends to be complicated and not easy for application administrators to use. Agrawal et al. [1] propose an XML-based language, called Autonomic Computing Policy Language (ACPL), to describe policies for autonomic computing. ACPL is designed as a general-purpose policy language. For example, it provides `Condition` and `Action` elements to describe a condition and an action to take. It can describe any types of policies, but not specialized to certain mechanisms. As well, [7] allows describing pairs of an environment

condition and an action through the use of general-purpose textual policy language. The proposed visual DSLs make it easy to understand, define, and maintain policies (i.e., agent behavior policies) in autonomic applications rather than general-purpose policy languages.

There are several DSLs to model biological systems such as biochemical networks for simulating and understanding biological systems (e.g., [10, 16]). However, the objective of the DSLs in iNetLab is different from theirs; DSLs in iNetLab aim to model biological (immunological) mechanisms for building autonomous and adaptive network applications. This work is the first attempt to investigate a DSL for biologically inspired autonomic networking.

J2EEML is a DSL to visually configure quality of service requirements and properties in Enterprise Java Beans (EJB) applications such as response time and message scheduling algorithms [25]. It assumes a stable domain-specific metamodel, and do not address the issue of customization of DSLs, i.e., do not provide means to customize metamodels. In iNetLab, application administrators not only use DSLs to model policies, but also customize DSLs through using DSLs. This mechanism allows even application administrators to customize DSLs to reflect the changes in the semantics of domain concepts.

A model transformation from a lower level (e.g., model) to a higher level (e.g., metamodel) is called *promotion* in the area of model-driven development. Similar to this work, [12] leverages this technique to create a new domain-specific metamodel from a model. However, [12] uses a general-purpose modeling language to describe a model to be *promoted* to a metamodel. In contrast, iNetLab uses DSLs to customize other DSLs. It simplifies the customization of DSLs and allows even application administrators to customize DSLs.

Several research efforts have investigated automatic generation of operational policies to satisfy desirable performance requirements [2, 15]. These techniques assume that each application's performance model is known. For example, several queuing models have been studied as the performance models for web servers. In those performance models, it is well-known how parameters (e.g., queue length) impact a web server's performance such as the average processing time for each incoming message. It is straightforward to estimate an application's performance when its performance model is known. In contrast, iNetLab does not assume any performance models, and its performance estimators are designed to approximate an application's performance without any prior knowledge on the applications. The estimation methods in iNetLab can be applied to autonomic applications whose performance models are not known.

8 Conclusion

iNetLab is a model-driven development and performance engineering environment to aid designing, maintaining and tuning operational policies in autonomic network applications. It provides (1) a set of DSLs to define operational policies in

autonomic network applications, (2) a set of supporting facilities for the DSLs, and (3) performance estimators to approximate the performance of an application using a certain behavior policy. With their supporting facilities, the proposed DSLs allow application administrators (i.e., non-programmers) to visually design and maintain operational policies in an intuitive manner. The proposed performance estimators can predict whether an application satisfies desirable performance requirements without running the application. This contributes to alleviate trial-and-error burdens in tuning behavior policies and shorten the time to develop autonomic applications.

References

1. Agrawal, D., Lee, K.W., Lobo, J.: Policy-based management of networked computing systems. *IEEE Communications Magazine* **43**(10), 69–75 (2005)
2. Bahati, R.M., Bauer, M.A., Vieira, E.M.: Policy-driven autonomic management of multi-component systems. In: *IBM International Conference on Computer Science and Software Engineering*, pp. 137–151. Ontario, Canada (2007)
3. Berek, C.: Somatic hypermutation and b-cell receptor selection as regulators of the immune response. *Transfusion Medicine and Hemotherapy* **32**(6), 333–338 (2005)
4. Bonacich, P.: Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology* **2**, 113–120 (1972)
5. Borgatti, S.P.: Centrality and network flow. *Social Networks* **27**(1), 55–71 (2005)
6. Chase, J., Anderson, D., Thakar, P., Vahdat, A., Doyle, R.: Managing energy and server resources in hosting centers. In: *ACM Symposium on Operating Systems Principles*, pp. 103–116. Banff, Canada (2001)
7. Dubus, J., Merle, P.: Applying OMG D&C specification and ECA rules for autonomous distributed component-based systems. In: *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Workshop on Models@Runtime*, pp. 242–251. Genova, Italy (2006)
8. Eclipse Foundation: Eclipse graphical modeling framework project. <http://www.eclipse.org/gmf/>
9. Eclipse Foundation: Eclipse modeling framework project. <http://www.eclipse.org/emf/>
10. Hucka, M., Finney, A., Bornstein, B., Keating, S., Shapiro, B., Matthews, J., Kovitz, B., Schilstra, M., Funahashi, A., Doyle, J., Kitano, H.: Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (sbml) project. *IEE Systems Biology* **406**, 41–53 (2004)
11. Jerne, N.K.: Idiotypic networks and other preconceived ideas. *Immunological Review* **79**, 5–24 (1984)
12. Jouault, F., Bézivin, J.: KM3: A DSL for metamodel specification. In: *IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, pp. 171–185. Bologna, Italy (2006)
13. Kasinger, H., Bauer, B.: Towards a model-driven software engineering methodology for organic computing systems. In: *IASTED International Conference on Computational Intelligence*, pp. 141–146. Alberta, Canada (2005)
14. Kephart, J.O.: Research challenges of autonomic computings. In: *ACM/IEEE International Conference on Software Engineering*, pp. 15–22. St. Louis, MO, USA (2005)
15. Kephart, J.O., Walsh, W.E.: An artificial intelligence perspective on autonomic computing policies. In: *IEEE International Workshop on Policies for Distributed Systems and Networks*, pp. 3–12. Yorktown Heights, NY, USA (2004)

16. Kolpakov, F.: Biouml – framework for visual modeling and simulation biological systems. In: International Conference Bioinformatics of Genome Regulation and Structure. Novosibirsk, Russia (2002)
17. Lee, C., Wada, H., Suzuki, J.: Towards a biologically-inspired architecture for self-regulatory and evolvable network applications. In: Advances in Biologically Inspired Information Systems, pp. 21–45. Springer (2007)
18. Lupu, E.C., Sloman, M.: Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering* **25**(6), 852–869 (1999)
19. openArchitectureWare.org: openarchitectureware. <http://www.openarchitectureware.org/>
20. Peña, J., Hinchey, M., Sterritt, R., Cortés, A., Resinas, M.: A model-driven architecture approach for modeling, specifying and deploying policies in autonomous and autonomic systems. In: IEEE International Symposium on Dependable Autonomic and Secure Computing, pp. 19–30. Indianapolis, IN, USA (2006)
21. Rohr, M., Boskovic, M., Giesecke, S., Hasselbring, W.: Model-driven development of self-managing software systems. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Workshop on Models@Runtime, pp. 115–116. Genova, Italy (2006)
22. Scott, J., Ideker, T., Karp, R.M., Sharan, R.: Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology* **13**(2), 133–144 (2006)
23. Trencansky, I., Cervenka, R., Greenwood, D.: Applying a UML-based agent modeling language to the autonomic computing domain. In: ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, onward! track, pp. 521–529. Portland, OR, USA (2006)
24. Wada, H., Lee, C., Suzuki, J., Otani, T.: A model-driven development environment for biologically-inspired autonomic network applications. In: IEEE International Workshop on Modelling Autonomic Communications Environments, pp. 25–41 (2007)
25. White, J., Schmidt, D., Gokhale, A.: Simplifying autonomic enterprise java bean applications via model-driven engineering and simulation. *Journal of Software and System Modeling*, Springer **7**(1), 3–23 (2008)

Network Reconfiguration in High-Performance Interconnection Networks

R. Casado, A. Bermúdez, A. Robles-Gómez, O. Lysne, T. Skeie, Å.G. Solheim, and T. Sørdring

Abstract High-performance interconnection networks such as InfiniBand, ASI, Autonet, and Myrinet have the property of link level flow control. As such, altering the routing function while the network is operational is a complex process, because the danger of packet deadlocks must be addressed. Over the past 5–6 years this problem has been addressed and solved by the research community.

This chapter collects several proposals for updating a routing function in interconnection networks after the occurrence of a topological change. Both traditional and recent reconfiguration schemes are covered, and the selection includes both technology-dependent and generic techniques.

1 Why Reconfiguring High-Performance Networks?

Modern high-performance networks that run real-time and bandwidth-intensive applications necessitate a communication subsystem architecture that not only ensures high levels of performance but must also sustain high robustness and availability. To support this, the network subsystem must be able to deal with both voluntary and involuntary changes in network state without experiencing significant performance degradation, and in application space this implies that user applications continue to run unimpeded and without termination.

The network must be able to adapt its state to ensure user requirements are met, that is it must remain functional in the presence of an altering topological state. The system may, for example, be required to sustain a hot-swap of components, partition the network based on aggregation, disaggregation, or for isolation, support the real-time activation and deactivation of nodes and switches as well as handling the introduction, removal, and failure of links and switches.

Reconfiguration of the network can be done either statically or dynamically. If the networking subsystem is implemented on a static basis, the detection of an anomaly or change requires a network shut-down and restart to establish and calculate

R. Casado (✉)

Universidad de Castilla-La Mancha, I3A Campus Universitario s/n, 02071 Albacete, Spain
e-mail: rcasado@dsi.uclm.es

connectivity and routing. A check-pointing algorithm is typically employed in user-space and an application rolls back to the last known correct state before operation continues. As the size of networks increases, the use of static models becomes undesirable due to the time required to re-initialize.

If a networking subsystem employs a dynamic model, the presence of an anomaly or change is met without termination of the application. The network continues operation as before but with perhaps a slight decrease in performance. This is achieved by either building intelligent logic into the switching components or by using a dynamic network reconfiguration strategy.

Any reconfiguration in the network must ensure that the network does not enter a state of deadlock—before, during, and after reconfiguration has taken place. Deadlock is an anomalous (permanent) state for a network to be in as either the whole network or parts of it have packets that can not make further progression because there are packets in front blocking them. Deadlocks are a by-product of the fact that modern day interconnection networks are loss-less, that is packets are not discarded during operation, even if the network is in saturation. Link-level flow control schemes (e.g., credit-based or Xon/Xoff) are normally used to ensure that a packet is not transmitted onto a (physical or virtual) channel unless there is sufficient buffer-space available for it to be stored on the receiving side of the channel.¹ The same holds for when a packet traverses from one channel to another (e.g., over a switch), there must be space available for the packet storage. This relation is commonly referred to as a *channel dependency*: If a packet may use a channel c_j immediately after a channel c_i there is a channel dependency from c_i to c_j .

Since no packet can advance until space is available in a receiving buffer, the risk of deadlock is inherent. A deadlock is a situation where, for a set of packets S , all the packets in S are waiting for another packet in S to progress before being allowed to advance itself. This progression denial is indefinite and based on a circular hold-and-wait dependency relation between network resources. To alleviate this situation, the networking subsystem would have to detect and resolve these deadlocked dependencies.

If a high-performance network employed a routing strategy that required deadlock detection and deadlock eradication, then the networks availability, reliability and performance would become unpredictable. Normally, the routing function of a loss-less interconnection network is carefully designed in order to ensure deadlock-freedom. Traditional graph theory is commonly used when analyzing the deadlock issue. All channel dependencies of a network are mapped to a *channel dependency graph*—a directed graph where each vertex represents a channel and each edge represents a dependency from one channel to another channel. According to [5] a deterministic routing function ensures deadlock-freedom if the channel dependency graph is free from cycles.

After a topology change within a network, there is a high likelihood that the routing function needs to be recalculated. Some topology changes can lead to major

¹ We assume that switches use input and output buffering.

routing recalculation while others may cause minimal or no recalculation, for example the loss of a leaf-node in an up*/down* [17] routed network.

A reconfiguration process exhibits four distinct phases; *detection*, *routing calculation*, *routing distribution*, and *new-routing activation*. The detection phase is handled by the hardware components within the network and can be in the form of, e.g., interrupts or polling. Once a state change is detected, the networking subsystem enters a routing calculation phase and determines the connectivity within the network as well as the routing function. Next, during the distribution phase, the networking subsystem distributes the routing information to the appropriate nodes within the network (switches for distributed routing networks and endnodes² for source routing networks). Later we will see there are also proposals where endnodes themselves calculate the routing function. In the final phase, known as activation, the new routing function replaces the old one and is activated. Dynamic reconfiguration strategies support activation of a new routing function while there are still packets belonging to the old routing function in the network. It is well known that, even if both the new and the old routing functions are deadlock-free, there is no guarantee that the combination of two deadlock-free routing functions results in a global deadlock-free routing function during the reconfiguration period. The concurrent mixture of packets belonging to the old and new routing functions allows for continued operation with minimal disruption to applications but introduces the potential of deadlock, which must be avoided. The potential for deadlock is a result of the introduction of transient (temporary) dependencies between network resources and falls away once there are no further packets belonging to the old routing strategy within the network.

One of the most crucial properties of an interconnection network is its topology; that is the structure of how the switches, links, and processing nodes are interconnected. The topology may be either regular or irregular. Typically, massively parallel and cluster computers are structured in a regular manner as the routing can then be optimized (by dedicated routing algorithms), to obtain the highest performance (when compared to topology agnostic routing algorithms). Examples of prominent topologies are meshes, tori, and multistage interconnection networks. The well known routing algorithm, dimension-order routing (DOR), has been around since the 1970s and is commonly used for routing within meshes and tori. The algorithm is easily implemented and its properties have been extensively studied. In a 2D mesh network, DOR is implemented either by traversing the horizontal dimension first followed by the vertical dimension (DOR-XY) or the vertical dimension first followed by the horizontal (DOR-YX).

Another prominent routing strategy is the up*/down* routing methodology, which is a deadlock-free routing algorithm that can be implemented on any topology (both regular and irregular). It is based on the assignment of a direction to each link within the network, configuring the topology as an acyclic directed graph with a single root node. To avoid deadlock, legal routes never use a link in the *up*

² An endnode is a compute node that generates, injects, receives, and processes data packets.

direction after having used one in the *down* direction. System area networks (inside a computer) may typically be structured irregularly and therefore require a generic routing algorithm. Note also that regularly structured networks may become irregular as soon as network components (switches and links) fail. Since topology specific routing algorithms have limited fault-tolerance capabilities, topology agnostic methods may also be deployed on regular topologies together with a dynamic network reconfiguration strategy to provide resilience.

In recent years we have seen the development of a class of topology agnostic routing strategies that use virtual channels to avoid deadlock and that exhibit performance that is almost as good as the topology specific algorithms. LASH (LAYERed SHortest path routing) [13] is an example of such a methodology where the shortest path between nodes are divided into virtual layers such that each layer is deadlock free, resulting in a global deadlock-free routing function. LASH guarantees shortest path routing requiring only a modest number of virtual channels.

The rest of the chapter is structured as follows. Section 2 revises the static reconfiguration approaches implemented by several high-performance interconnection technologies. Section 3 provides a theoretical framework for dynamic reconfiguration. In Sects. 4–6, several generic dynamic reconfiguration techniques are detailed. Finally, in Sect. 7 we summarize the properties of the different reconfiguration techniques.

2 Static Reconfiguration Approaches

In this section, we present some high-performance interconnection networks that can handle topological changes. In recent years, some mechanisms based on static reconfiguration have been proposed for these kind of technologies.

2.1 *Autonet*

Autonet [17] used a distributed up*/down* routing scheme, and its reconfiguration mechanism statically updated the forwarding tables in each switch. When one or more nodes detected a change, a process consisting of three phases was initiated.

1. The first phase consists of the construction of a minimum depth spanning tree (MDST) for the network. Each node detecting the topology change generates a new tree and notifies its neighbors, who decide either to join that tree or begin the construction of a new one (if they have a lower identifier). The node executes the algorithm shown in the Fig. 1. At the end, the minimum spanning tree spreading from the node with the lowest identifier remains in the network. Figure 2 shows an example of the computation of a spanning tree. When a node determines that it is a leaf (i.e., none of its neighbors is connected to the tree through this node), it concludes that the process has terminated, and notifies its parent of its topology information. Each node in the tree waits for topology information from all its

Fig. 1 Algorithm executed for the construction of a network spanning tree. During the process, nodes may migrate between several trees, as well as change their position in the tree

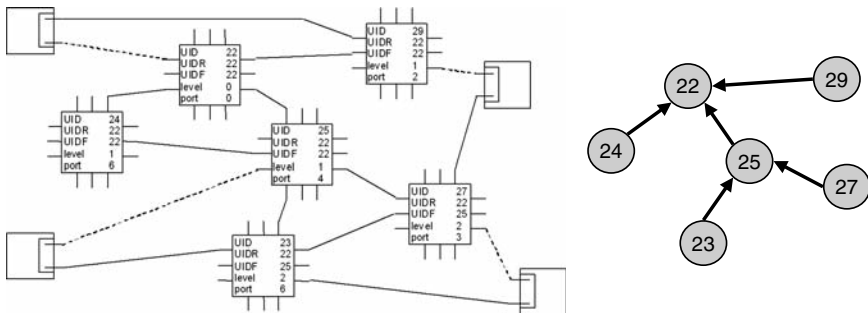
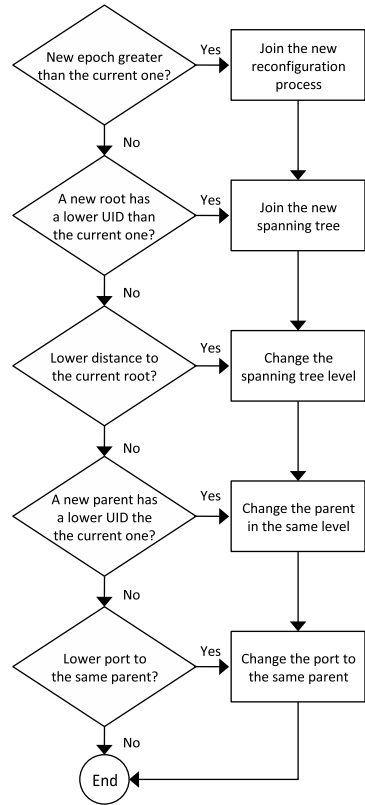
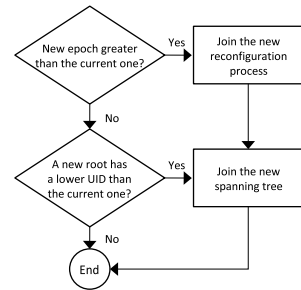


Fig. 2 (left) Network topology including six switches and four hosts. Switch ports are numbered from 1 to 9 in clockwise direction, starting from the left bottom port. Each switch contains a unique identifier (UID), the spanning tree root UID (UIDR) and level (prof), the tree parent UID (UIDF), and the port connecting to it (port). (right) Graphical representation of the resulting spanning tree

Fig. 3 Algorithm executed for the construction of a propagation-order spanning tree (POST)



children before recursively communicating it to its parent. The first time a switch receives a control packet indicating reconfiguration, it resets its forwarding tables and starts discarding application traffic.

2. The second phase of the process consists of the distribution of the entire network topology from the root node to all the switches throughout the spanning tree.
3. Finally, in the third and final phase, each switch computes and loads its own forwarding table, based on the information received, and starts accepting application traffic.

It is possible that a new reconfiguration process starts before the previous one has finished. To manage simultaneous reconfigurations, Autonet incorporates an epoch mechanism based on assigning a unique value to each reconfiguration. In this way, switches abandon obsolete reconfiguration processes and only join a new process.

Rodeheffer and Schroeder [16] replaced the MDST by a propagation-order spanning tree (POST). As a consequence, nodes do not change between parents in the same tree in order to reduce the depth of the tree, as shown in Fig. 1. Moreover, the final root is the node with the lowest identifier that detects the change. This algorithm, shown in Fig. 3, converges faster than the previous one, reducing the time the network is not operational.

2.2 Myrinet

Myrinet [3] is a high-performance interconnection network based on source routing. The initial reconfiguration approach applied to this technology was a deadlock recovery scheme. When a packet is not completely received before a certain timeout expires, the packet is discarded.

In a second version, Myricom decided to incorporate a static reconfiguration approach similar to the one implemented by the Autonet network. The process is centralized in an entity—the *mapper*—executed in the interface of a host. The mapper is in charge of detecting the occurrence of a topological change. In particular, it periodically starts an exploration process in order to obtain a map of the topology. This process sends exploration packets that start at the mapper, reach a certain network node, and return to the mapper using the same path in the opposite direction.

Each time the mapper discovers a new switch, it sends an exploration packet through each port of that switch, assuming that there is a device at the other side of the port. If a packet returns to the mapper, this means that this assumption was right. Once the mapper has collected the complete map, it is distributed to each host in the network. From this information, each host computes its own set of routes.

2.3 InfiniBand

The first proposals to statically reconfigure InfiniBand networks were similar to those described in the previous sections for Autonet and Myrinet networks [2]. In short, all network ports are (logically) deactivated, that is, they do not accept application packets. Then, forwarding tables can be sent to network switches. Finally, the network ports are reactivated and application traffic is again accepted.

In [1] two alternative pseudo-static proposals based on up*/down* routing are described. The key idea is to relax the traditional static reconfiguration restrictions, either by reducing the amount of network ports that are deactivated, or by only preventing certain port transitions while the routing function is being updated.

The first proposal modifies the first step of the traditional static reconfiguration process such that only the ports of the switches that act as break nodes³ in the new up*/down* directed graph are deactivated. Moreover, it is not necessary to deactivate all ports in those nodes. Instead, it is enough to select those ports that are sources of *up* links, and deactivate all except one of them. This ensures that the forbidden transitions in the new configuration will not be used by any packet using the old routing function. Figure 4 shows an example.

This simple mechanism reduces the negative effects of a “pure” static reconfiguration process. However, there are still many paths that could be used during the distribution of tables, without introducing potential deadlock situations. The process can be improved upon by preventing the use of a few input port–output port combinations, in particular those corresponding to the forbidden transitions at the new break nodes. This proposal does not require deactivation of network ports. For the example, in Fig. 4, it is only necessary to prevent the forbidden transitions at nodes 6 and 10 while the new forwarding tables are being distributed.

Figure 5 shows the effects of the static and pseudo-static reconfiguration techniques on application traffic, for an irregular subnet comprised of 24 switches and 22 hosts and a change consisting of the addition of a switch. The plots show how the optimized techniques improve network throughput during the reconfiguration process. Static reconfiguration has a detrimental effect on instantaneous throughput. It is clear that the pseudo-static techniques improve considerably on this behavior. In fact, when the technique based on the deactivation of break node dependencies is used, the gap representing reduced performance that was visible in the bottom plot completely disappears.

³ A break node is a switch that has several outgoing upward links.

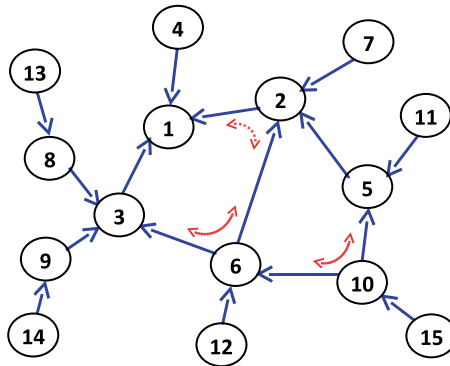
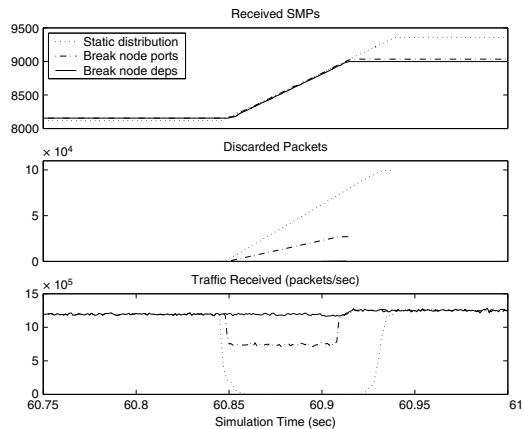


Fig. 4 Pseudo-static reconfiguration based on port deactivation. Assume the previous break node in the cycle on the left was the node labeled as 2. That means that the direction assigned to the link connecting nodes 6 and 2 in the previous configuration was the opposite of the currently assigned direction. In this situation, the reconfiguration process needs only deactivate one of the ports in node 6 that are the sources of *up* links (connecting either node 2 or 3). Similarly, it is only necessary to deactivate one of the ports of node 10 that are sources of *up* links (connecting either node 5 or 6)

Fig. 5 Instantaneous aggregated received management packets (SMPs), discarded packets, and traffic received versus simulation time for an irregular network comprising 24 switches and 22 hosts. A reconfiguration starts at time 60.85



3 Dynamic Reconfiguration Principles

In Sect. 1, we stated that in interconnection networks with link level flow control, care must be taken so that the network does not enter a deadlocked state. We also discussed some theoretical aspects that form the basis of how a routing function can guarantee freedom from deadlock for a network. As such it is tempting to assume that deadlock-free dynamic network reconfiguration can be achieved by simply guaranteeing that the new routing function has a cycle-free (escape) channel dependency graph at all points in time. Unfortunately this is insufficient.

A previously active routing function continues to exercise its presence if there exist some packets within the network that occupy channels (routed according to

the old routing function) that they would not occupy if they were routed according to the current (new) routing function. This leads to two problems, the first of which is called *ghost dependencies*. These are dependencies that do not stem from the current routing function, but are there as the effect of a previous routing function having placed packets in places that they would otherwise not be. The second problem is that of *unroutable packets*, i.e., packets that cannot be routed because they are placed in positions (buffers and channels) where the new routing function does not expect them to be.

Any mechanism for dynamic reconfiguration has to tackle both ghost dependencies and unroutable packets. This is usually done through synchronization points, that allow some packets to drain from certain channel queues before routing options that may cause these effects are removed/introduced.

A theory for reasoning about deadlock freedom for dynamic reconfiguration was developed in [8]. Unlike previous theories (e.g., [5–7, 9]), this theory encompasses deadlock freedom for networks which undergo reconfiguration. The theory is based on describing network states, i.e., the placement of packets within the network, as sets of *CND-tuples*.⁴ Each tuple describes the occupancy of a network channel and possible dependencies to other channels for a packet destined to a particular endnode. For a routing function R , $R(c, d)$ is the set of channels that a packet header residing in channel c can take for a destination given by d . With this, CND-tuple $\langle c, c', d \rangle$ signifies that channel c is occupied by part of a packet destined to endnode d whose header was routed to channel c' . A CND-tuple $\langle c, c', d \rangle$ is *legal* for R only if $c' \in R(c, d)$. Alternatively, CND-tuple $\langle c, c, d \rangle$ signifies that channel c is occupied by part of a packet destined to endnode d whose header has not yet been forwarded further to another channel.

According to [8], the runtime state during reconfiguration is represented as a *CND-relation*, which is a set of CND-tuples. The set of all possible runtime states at a given stage of a reconfiguration process is represented by a superset of CND-tuples. That set is legal if it is the union of all the CND-relations that represent all legal runtime states at a given point in the reconfiguration process. If Θ is a set of legal CND-tuples that represents a set of legal runtime states, any given one of these runtime states is represented by a CND-relation formed from a consistent subset of Θ .

Let us now redefine some of the concepts that were used to study deadlocks for static routing functions. The first concept that we study is that of a channel dependency. When the routing function is undergoing changes, we can no longer deduce deadlock from the routing function alone, because another routing function that was previously active may have placed packets into various positions within the network that cannot be deduced from the current routing function. The channel dependency concept must therefore also take into account a configuration of existing

⁴ As described in [8], CND is short for *Current channel*, *Next channel*, and *Destination*. Several CND-tuples are needed to represent a blocked wormhole packet that spans multiple channels, whereas a single tuple is needed to represent a blocked virtual cut-through packet given that channels are able to store entire packets.

packets within the network, as specified by a CND-relation, as well as the current routing function R .

Intuitively, we need to define that there is a dependency from channel c to channel c'' if the *previous* routing functions may have left the network in a state where the header of the packet that currently occupies c may use c'' as its next buffer according to the *current* routing function. This will maintain the connection between deadlocks and cycles in the graph of channel dependencies, because we have only generalized the concept of dependencies according to a more complex picture of where packets may reside. In the following we let $Header(c)$ denote the channel that holds the header of the packet that occupies channel c , and we let P denote the set of possible destinations for packets.

Definition 1 For any current routing function R and set of CND-tuples Θ , there is a channel dependency from channel c to channel c'' if there exists a channel c' and a node $d \in P$ such that $c' = Header(c)$ in some consistent subset θ of Θ and $c'' \in R(c', d)$. By $DEP(R, \Theta)$ we denote the set of channel dependencies corresponding to routing function R and a set of CND tuples Θ .

We now turn our focus to the concept of completeness of subsets of channel dependencies. This concept will later be used to define properties of escape channel sets. For static routing functions this concept captures the ability of the escape channels to move all packets toward their destinations. Again we will have to adjust this notion to cater for a situation where packets may reside in places where the prevailing routing function would not have put them. We now need to define that a subset of channel dependencies is complete if all packets in any possible configuration of packets in the network has at least one routing choice that is reflected into one dependency in the set. This maintains the property that all packets can reach its destination by using channels in the order defined by dependencies in a complete subset.

Definition 2 For any routing function R and set of CND-tuples Θ , a subset of channel dependencies $\Delta \subseteq DEP(R, \Theta)$ is complete if $\forall c, c' \in C$ and $\forall d \in P$ for which $c' = Header(c)$ for some consistent subset θ of Θ and $\langle c', c', d \rangle \in \Theta$, there exists $c'' \in R(c', d)$ such that $\langle c, c'' \rangle \in \Delta$.

At this point, the notion of escape channel dependency sets is easy to define:

Definition 3 For any routing function R and set of CND-tuples Θ , a subset of channel dependencies $\Delta \subseteq DEP(R, \Theta)$ is an escape channel dependency set if Δ is complete and is an acyclic relation on C .

Now we are ready to state the main theorem of this section. For the complete proof, we refer to [8].

Theorem 1 For any reconfiguration process RP consisting of k steps, let R_k be the target routing function of RP and let a set of legal CND-tuples Θ_{final} represent the class of legal configurations when RP finishes. The network is deadlock-free after reconfiguration finishes if there is an escape channel dependency set $\Delta \subseteq DEP(R_k, \Theta_{final})$.

Note that this theorem takes only the final state of the reconfiguration process into consideration. Its main statement is that if the network is deadlock free at the end of the reconfiguration process, then the entire reconfiguration process can be considered deadlock free. Any transient deadlocks during reconfiguration will have been resolved, and any unroutable packet will again be routable. It is, however, important to note that this formulation of the theorem assumes that the final stage of reconfiguration is actually reachable. This assumption is not trivial, since it requires, e.g., that control messages are not trapped in transient deadlocks. If the control messages are transmitted out of band, this will be easy to guarantee. If they are transmitted in-band, we will need to guarantee that the reconfiguration process is also free from transient deadlocks. For a further study of this topic, we refer to [8].

The way this complexity is handled in the reconfiguration processes we describe below is basically that the class of legal packet configurations is controlled by synchronization steps, limiting the set of ghost dependencies and unroutable packets in the network to a controlled and tractable set. This is most obvious in static reconfiguration, where all packets are drained from the network before the routing algorithm is changed. Clearly this will remove all ghost dependencies, thus static reconfiguration will be deadlock free when reconfiguration is finished and packet injection is resumed. The careful reader will in all reconfiguration methods we describe below find similar synchronization steps where the process is halted awaiting special ghost dependencies to disappear.

4 Reconfiguration Based on Evolving Graphs

For a given network topology, up*/down* routing is based on a link direction assignment that represents a correct graph, that is an acyclic directed graph with only one root node. When some switches are added to or removed from the network, the topology changes, and the up*/down* graph may become incorrect (due to the existence of either several root nodes or routing cycles). Some reconfiguration proposals are based on evolving such an incorrect graph into a correct one. A seminal technique based on this idea was Partial Progressive Reconfiguration (PPR) [4]. Recently, a more computationally efficient approach was presented [15]. We discuss both approaches in the following.

4.1 Partial Progressive Reconfiguration

PPR was proposed for distributed routing networks that are able to asynchronously update the routing tables without stopping application traffic. When a new switch is added, a direction must be assigned to the links connecting to it. This assignment should not produce cycles in the directed graph. A simple approach consists of assigning a direction to those links in such a way that the *down* direction goes toward the new switch. By doing so, messages will be able to use the new links

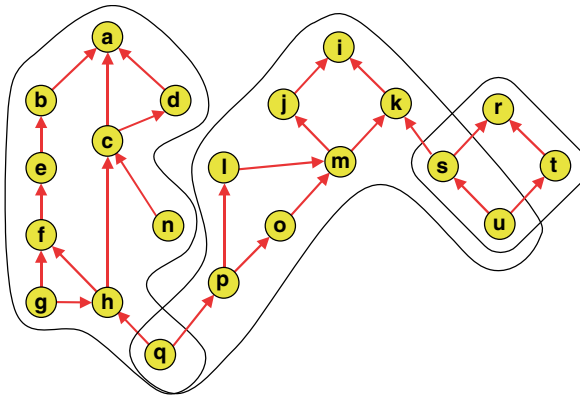


Fig. 6 Incorrect graph decomposed into correct regions

to route to/from the new switch, but not to cross it. If the new switch is connected to the network through two or more links, it will become a break node (or a false break node). On the other hand, switch deactivations cannot produce cycles in the directed graph, but they may produce the appearance of several root nodes, as shown in Fig. 6.

When a directed graph contains several root nodes, it is not possible to route messages between them. In this case, it is possible to split the directed graph into several correct subgraphs, called *correct regions*. In fact, the network has as many correct regions as root nodes (one root node in each correct region). Figure 6 shows the correct regions inside the incorrect graph.

Certain switches, called *frontier nodes*, have all their upward links crossing the limits of any region containing them. Between every pair of overlapping regions, the reconfiguration algorithm selects one frontier node as *router node*. In the graph in Fig. 6 nodes *q* and *s* are router nodes.

All root and router nodes execute a distributed protocol to generate a spanning tree. As a result, only one root node remains in the tree. The others should change their link directions to the respective router nodes connecting the primary root node. Only the orientation of the links connected to the root or break nodes can be changed, otherwise, some nodes may be unreachable. Taking this into account, each secondary root node begins its movement, exchanging its position with a neighboring node; the current node exchanges its position with the following one, and so on.

A moving root node cannot move over a break node because a cycle arises. With this restriction, it may happen that there is no valid path between the secondary root node and the router node that must be reached. In Fig. 6, node *a* must reach node *q* but break nodes *c*, *g*, and *h* prevent it. The solution consists of previously moving the node *h*, keeping it away from the path followed by the secondary root node. Figure 7 shows the situation of Fig. 6 after all the movements have been performed.

The movement of a break node toward its neighbor may cause deadlocks if it is done in an uncontrolled way. To solve this, PPR assures the complete deactivation of old channel dependencies by redirecting traffic over the other part of the cycle (being

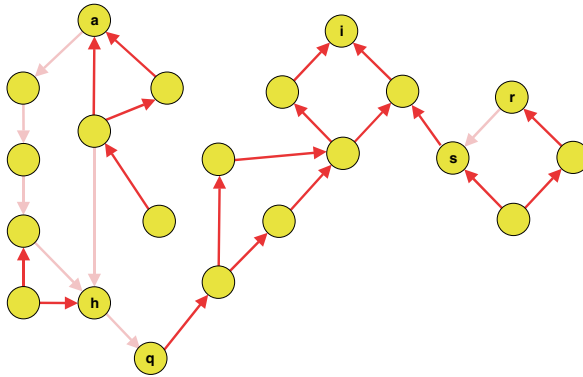


Fig. 7 Correcting the real graph

split into multiple paths in case of multiples cycles). After that, new dependencies are activated redirecting part of the traffic through them again.

Next, we compare the performance of PPR with POST. In Fig. 8 we can see that the average packet latency is approximately 3,000 cycles in the absence of reconfiguration. For a simulation time equal to 510×10^6 cycles, a reconfiguration process is triggered. A POST reconfiguration produces a time interval of half a million cycles in which packets are not transmitted through the network. The absence of packets is a result of prohibiting application traffic during the reconfiguration. Moreover, more than 3,000 application packets are discarded. On the other hand, a PPR technique does not affect application traffic. The network does not stop its activity, and packet latency does not increase during the reconfiguration. Only a few packets are discarded by the PPR process.

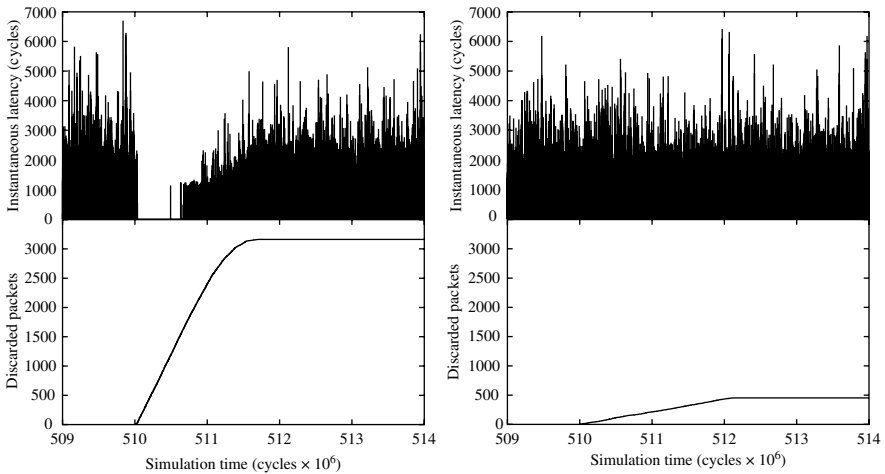


Fig. 8 Instantaneous latency and aggregated discarded packets versus simulation time for an irregular network composed of 48 switches, using (left) POST and (right) PPR

4.2 Close Graph-Based Reconfiguration

In [15] a recent approach, Close Graph-based Reconfiguration (CGR), was presented. It is applicable to distributed as well as source routing networks. As opposed to PPR, the graph-evolving process is performed in a centralized fashion. Additionally, certain restrictions are imposed on the construction of the new up*/down* graph. In particular, a break node in the old graph remains a break node in the new graph or the role of break node is transferred to a node at most one hop away. We say that the new graph is close to the old one.

The restrictions above ensure that new packets cannot perform transitions that are prohibited by the old routing function, and that old packets cannot perform transitions that are prohibited by the new routing function. The new routing function is deadlock free and ensures network connectivity. Moreover, packets routed according to the old and the new routing functions can unrestrictedly coexist in the network, without the risk of forming deadlocks [15]. This allows for an updating of the routing function in a completely dynamic way.

Figure 9 illustrates the instantaneous behavior of the CGR scheme (comparing it with POST). For both schemes, we have scheduled a removal of a switch in a 6×6 mesh at time 2.0s. For all plots, the x -axis represents the simulation time. Simulation results show that the CGR technique significantly reduces the amount of packets that are discarded during the topology-change assimilation. From the point of view of upper-level applications, the new reconfiguration strategy virtually eliminates the problem of reduced network service availability. In addition, the proposed strategy does not require additional fabric resources such as virtual channels, and it could easily be implemented in current commercial systems.

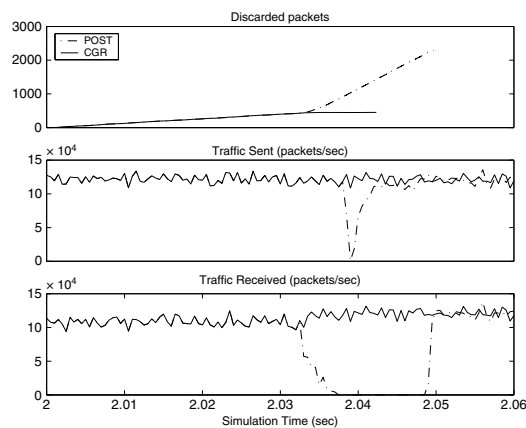


Fig. 9 Instantaneous aggregated discarded packets and throughput versus simulation time for a 6×6 mesh. A reconfiguration starts at time 2.32

5 Double Scheme

Double Scheme [14] is a dynamic reconfiguration scheme that enables a transition between any pair of deadlock-free routing functions. It was designed for networks supporting virtual channels and adaptive routing with escape channels [10]. In [18] a method for applying Double Scheme over InfiniBand networks is proposed and evaluated.

In adaptive routing scenarios, packets normally use adaptive routing over one or more virtual channels. However, adaptive routing is not deadlock free. So, in order to avoid potential deadlocks, the network must support an additional escape channel, which must be used by a packet when there are no adaptive channels available at an intermediate switch. Packets in the escape channel are routed to a destination according to any deadlock-free routing function, as for example up*/down*.

The key idea behind Double Scheme is to spatially separate packets routed according to the routing function in use before reconfiguration (R_{old}) from packets routed according to new routing function (R_{new}) by doubling the escape channel resources during reconfiguration. One set of resources is used exclusively by R_{old} and the other by R_{new} . In this way, packets in each set are routed under the influence of one and only one routing function—either R_{old} or R_{new} , but never both. By allowing dependencies to exist from one set of resources to the other, but not both ways, at any given time, a guarantee of deadlock freedom during and after reconfiguration can be proved [14].

There are two main steps in the Double Scheme. First, one of the adaptive channels is drained and configured as the new escape resource. Afterward, the old escape resource is used as a new adaptive resource. In greater detail, let us assume that R_{old} allows packets to be injected into a connected set of routing resources (designated as C_{old}). Once R_{new} is computed, a connected set of routing resources (designated as C_{new}) is required for use by newly injected packets routed under the influence of R_{new} . Such resources could be made available by allowing packets in a subset of the C_{old} set of resources to be delivered to their destinations while stopping the injection of new packets into that subset, which essentially drains those resources. As packets are no longer injected into any of the C_{old} resources after the C_{new} resources are taken into use, the C_{old} resources eventually become free and can be incorporated into the set of C_{new} resources once completely empty.

6 Overlapping Reconfiguration

Overlapping Reconfiguration [11, 12] accepts application traffic into the network during the transition from one deadlock-free routing function (R_{old}) to another (R_{new}), and uses a special packet called a *token* to ensure that the transition does not cause deadlock.

The packets routed according to R_{old} and R_{new} are denoted old packets and new packets, respectively. The main idea behind Overlapping Reconfiguration is to avoid

deadlocks during the transition from R_{old} to R_{new} by using the token to separate old packets from new packets on each (physical or virtual) channel, such that each channel first transmits old packets, then the token, and finally new packets. If this invariant is maintained, no new packets can block old packets and vice versa, and, as R_{old} and R_{new} are both deadlock free, deadlocks cannot form. Thus, unlike, e.g., Double Scheme, Overlapping Reconfiguration does not need virtual channels for deadlock avoidance.

In order to govern token and data packet forwarding, additional logic and state information are required in the switches, and each switch must know the dependencies from its input channels to its output channels. Overlapping Reconfiguration originally targeted only distributed routing systems, but an adaptation to source routing systems was recently proposed. Each packet must be routed from source to destination according to only one of the routing functions, and a distributed routing switch or a source routing endnode must hold routing tables for both R_{old} and R_{new} while the transition is accomplished. Thus, during the reconfiguration, a system's required amount of routing information doubles. Any routing algorithm (or pair of routing algorithms) can be used with Overlapping Reconfiguration.

The core of the Overlapping Reconfiguration algorithm is the requirement that each channel first transmits old packets, then the token, and finally new packets. In the following it is assumed that only one reconfiguration process is ongoing at a time, and that packets heading for a faulty channel are removed from the network. The algorithm consists of three main parts that state the responsibilities of the injection channel of an endnode, the input channel of a switch, and the output channel of a switch.

Endnode Injection Channel: Each injection channel of each endnode transmits a token to indicate that subsequent packets must be routed according to R_{new} .

Switch Input Channel: An input channel routes packets according to R_{old} until the token is processed (having reached the head of the input queue), and thereafter routes packets according to R_{new} . An input channel that has processed the token forwards packets solely to output channels that have already transmitted the token. Packets bound for output channels that have not yet transmitted the token are temporarily held back.

Switch Output Channel: An output channel, c_o , does not transmit the token until all input channels, c_i , for which channel dependencies exist according to R_{old} from c_i to c_o , have processed the token. If no such dependency exists for an output channel, it should transmit the token when the first token has been processed by any of the input channels.

During the transition from one routing function to another packets may experience increased delay due to the token forwarding regime. Tokens flow through the network in accordance with the channel dependency graph of R_{old} . Assume that a switch has channel dependencies from input channels c_{i0} , c_{i1} , and c_{i2} to output channel c_{o2} , that c_{i0} and c_{i2} have not yet processed the token, and that a new packet bound for c_{o2} arrives on c_{i1} (which has processed the token). The new packet cannot

be forwarded to c_{o2} until c_{o2} has transmitted the token, which depends on prior token processing by both c_{i0} and c_{i2} . Thus, the packet must temporarily be held back and experiences additional delay.

For an 8×8 mesh, 16×16 mesh, and 16×16 torus, Fig. 10 shows simulation results for a scenario where the reconfiguration is merely due to a change of routing function, there are no changes in topology. The reconfiguration is initiated after 8,500 cycles. The up*/down* routing algorithm is used to calculate both R_{old} and R_{new} . The root node is in the upper left corner of the topology for R_{old} and in the upper right corner for R_{new} (when ignoring the wraparound links of the torus). Two traffic load levels are in focus, corresponding to 60% (low load) and 90% (high load) of network saturation.

The reconfiguration period starts when the first token is injected by an endnode (in this case all endnodes inject their tokens simultaneously) and ends when the last token is received by an endnode. In Fig. 10(a) and (f), the start and end of the reconfiguration period are indicated by vertical lines. When the traffic load increases from low to high, the reconfiguration period is prolonged by almost 15% for the 16×16 mesh and 16×16 torus and by almost 5% for the 8×8 mesh.

The $Latency_{time}$ metric results from the division of the data collection period into 200 time intervals, each with a duration of 100 cycles (a cutout covering the reconfiguration period is presented). For a time interval int , $Latency_{time}$ is the average latency of all packets that are generated by any endnode in int and that subsequently reach their destination endnode. The latency for a single packet is the time that elapses from when the packet is generated and injected into the transmission queue of the source endnode until the packet is received by the destination endnode.

Figure 10(a) and (d) shows $Latency_{time}$ for an 8×8 mesh under a hotspot traffic pattern for low and high traffic load, respectively. For the hotspot traffic pattern

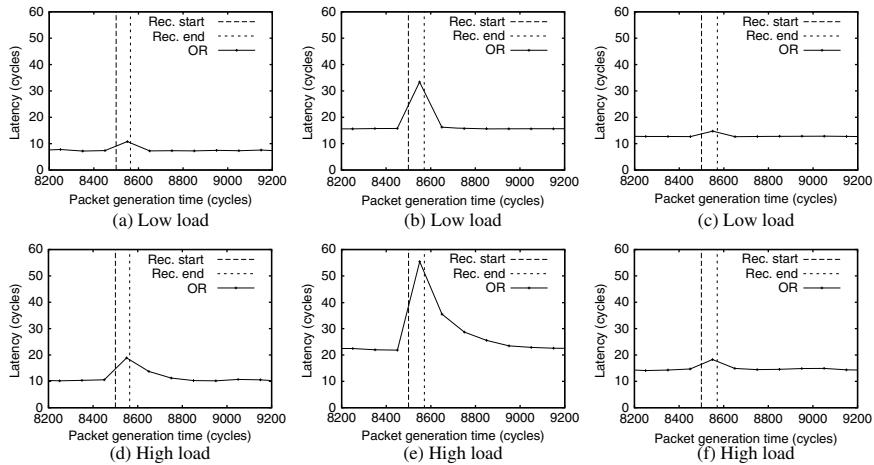


Fig. 10 $Latency_{time}$ for Overlapping Reconfiguration (OR) for an 8×8 mesh under hotspot traffic (a, d), 16×16 mesh under uniform traffic (b, e), and 16×16 torus under uniform traffic (c, f)

80% of the traffic is destined for a node in the middle of the mesh whereas the remaining 20% has a uniform destination address distribution. For a purely uniform destination address distribution, corresponding results are shown for a 16×16 mesh in Fig. 10(b) and (e) and for a 16×16 torus in Fig. 10(c) and (f). All the plots show crests in the $Latency_{time}$ curve due to the reconfiguration during which a number of packets are temporarily held back in the switches while awaiting token forwarding. As the traffic load increases, both the height and width of these crests increase for the 8×8 and 16×16 meshes. For the 16×16 torus this is not the case, however.

The ratios of the peaks of the $Latency_{time}$ curves to the latency values before reconfiguration starts are 1.5 and 1.8 (8×8 mesh), 2.1 and 2.5 (16×16 mesh), and 1.2 and 1.2 (16×16 torus) for low and high traffic load, respectively. The up*/down* graphs for the 16×16 torus and the 16×16 mesh are different. When based on a spanning tree identified by a breadth-first search as in [17], the up*/down* graph becomes shallower and wider for the torus than for the mesh. The simulation results indicate that, for the routing functions under study, the packet latency is more affected by the reconfiguration for the mesh than for the torus.

Overlapping Reconfiguration has been categorized both as a dynamic reconfiguration scheme [12] and as a static reconfiguration scheme with overlapping phases [11]. Application traffic is accepted into the network while the routing function is being updated, in this respect Overlapping Reconfiguration resembles a dynamic reconfiguration strategy. On the other hand, the use of tokens to separate the packets belonging to R_{old} and R_{new} on each channel suggests that Overlapping Reconfiguration could be categorized as a static reconfiguration scheme with overlapping phases. Overlapping Reconfiguration does not exhibit the detrimental effect on the network service that is typical of traditional static reconfiguration schemes. However, as with any generic reconfiguration scheme, some increase in packet latency must be expected.

7 Summary

This chapter has discussed the problem of updating a routing function after a topological change in high-performance interconnection networks. Traditional mechanisms use static reconfiguration in order to solve this problem; although, when this kind of reconfiguration is considered, large amounts of data packets are discarded due to the utilization of the obsolete routing function, but is primarily due to the deactivation of network ports. For this reason, these solutions have potentially several negative effects on network service availability.

With the aim of reducing or eliminating these potential negative effects, some proposals were developed in order to update the routing function dynamically. The most relevant dynamic reconfiguration mechanisms available in the literature have been described. When compared to the traditional solutions, these techniques significantly reduce the change assimilation time and the negative impact on the application performance.

Some of the dynamic reconfiguration techniques are tailored for use with the up*/down* routing algorithm. PPR is an efficient reconfiguration mechanism designed for distributed environments. Additionally, CGR is a recent technique based on the same principles as PPR. It obtains a new routing function which ensures that the packets routed according to the old and the new routing functions can unrestrictedly coexist in the network, without the risk of deadlocks forming. CGR could easily be implemented in current commercial systems using either source or distributed routing schemes.

Two dynamic reconfiguration schemes that can be used with any routing algorithm (or pair of routing algorithms) are Overlapping Reconfiguration and Double Scheme. Overlapping Reconfiguration uses a special packet to separate the data packets routed according to the old and the new routing functions on each physical or virtual channel. It was originally proposed for networks based on distributed routing and, recently, adapted for source routing environments. Also, for source routing systems, an optimization to the original algorithm was proposed that enables immediate forwarding of new packets that have valid routes according to the old routing function, regardless of the token forwarding status of a switch.

The previous mechanisms do not require additional network resources such as virtual channels. In contrast, Double Scheme uses virtual channels when reconfiguring the network in a deadlock-free manner. This scheme uses one set of virtual channel resources for the remaining packets that are routed according to the old routing function, and another set of virtual channel resources for the packets routed according to the new routing function. Double Scheme is not applicable in networks that have limited virtual channel resources.

To sum up, dynamic reconfiguration techniques significantly reduce, and in some cases avoid, the negative impacts that are observed when static reconfiguration is applied. When selecting a reconfiguration scheme, issues that should be considered include network topology, available resources within the network, the particular routing algorithm used, and the requirements of the application traffic in terms of latency and throughput.

Acknowledgments This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants “Consolider Ingenio-2010 CSD2006-00046” and “TIN2006-15516-C04-02”; by JCCM under grant “PBC05-007-1.” It has also been supported by a FPI grant under the Spanish MEC “TIC2003-08154-C06-02.”

References

1. Bermúdez, A., Casado, R., Quiles, F.J.: Distributing InfiniBand forwarding tables. In: EuroPar 2004 Conference, pp. 864–872. Pisa, Italy (2004)
2. Bermúdez, A., Casado, R., Quiles, F.J., Pinkston, T.M.: Evaluation of a subnet management mechanism for InfiniBand networks. In: 2003 International Conference on Parallel Processing (ICPP’03), pp. 117–124. IEEE Society Press (2003)
3. Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.: Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro* **15**(1) (1995)

4. Casado, R., Bermúdez, A., Duato, J., Quiles, F.J., Sánchez, J.L.: A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* **12**(2), 115–132 (2001)
5. Dally, W.J., Seitz, C.L.: Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers* **C-36**(5), 547–553 (1987)
6. Duato, J.: A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* **6**(10), 1055–1067 (1995)
7. Duato, J.: A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* **7**(8), 841–854 (1996)
8. Duato, J., Lysne, O., Pang, R., Pinkston, T.M.: Part I: A theory for deadlock-free dynamic network reconfiguration. *IEEE Transactions on Parallel Distributed Systems (TPDS)* **16**(5), 412–427 (2005)
9. Duato, J., Pinkston, T.M.: A general theory for deadlock-free adaptive routing using a mixed set of resources. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* **12**(12), 1219–1235 (2001)
10. Duato, J., Yalamanchili, S., Ni, L.: *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers (2002)
11. Lysne, O., Montañana, J.M., Flich, J., Duato, J., Pinkston, T.M., Skeie, T.: An efficient and deadlock-free network reconfiguration protocol. *IEEE Transactions on Computers* **57**(6), 762–779 (2008)
12. Lysne, O., Montañana, J.M., Pinkston, T.M., Duato, J., Skeie, T., Flich, J.: Simple deadlock-free dynamic network reconfiguration. In: 11th International Conference on High Performance Computing (HiPC 2004), pp. 504–515 (2004)
13. Lysne, O., Skeie, T., Reinemo, S.A., Theiss, I.: Layered routing in irregular networks. *IEEE Transactions on Parallel and Distributed Systems* **17**(1), 51–65 (2006)
14. Pang, R., Pinkston, T.M., Duato, J.: The double scheme: Deadlock-free dynamic reconfiguration of cut-through networks. In: *Proceedings of the 2000 International Conference on Parallel Processing (ICPP'00)*, pp. 439–448 (2000)
15. Robles-Gómez, A., Bermúdez, A., Casado, R., Solheim, Á.G.: Deadlock-free dynamic network reconfiguration based on close up*/down* graphs. In: *Euro-Par 2008 Conference* pp. 940–949 (2008)
16. Rodeheffer, T.L., Schroeder, M.D.: Automatic reconfiguration in autonet. In: *SRC Research Report 77 of the ACM Symposium on Operating Systems Principles* (1991)
17. Schroeder, M.D., Birrell, A.D., Burrows, M., Murray, H., Needham, R.M., Rodeheffer, T.L., Satterthwate, E.H., Thacker, C.P.: Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications* **9**(8), (1991)
18. Zafar, B., Pinkston, T.M., Bermúdez, A., Duato, J.: Deadlock-free dynamic reconfiguration over InfiniBand. *Special Issue on Parallel and Distributed Algorithms, International Journal of Parallel Algorithms and Applications (IJPA)* **19**(2–3), 127–143 (2004)

Autonomic Management of Networked Web Services-Based Processes

Farhana H. Zulkernine, Wendy Powley, and Patrick Martin

Abstract Web services, which have evolved along with the World Wide Web, hold great potential for building multi-organizational dynamic workflows. The management of these networked Web service-based processes presents significant challenges. The varying workload of the Internet, the heterogeneous multi-component structure of the Web services environment, and the complexity in building and managing Web services-based workflows place huge management problems upon the system administrators of the Web services environment and the Web services consumers. Autonomic computing has received considerable attention in the research community as a potential approach to making these systems self-managing. We discuss Web services management from two different perspectives, namely from those of the service provider and the service consumers. We present our approaches to autonomic management for both perspectives and discuss their integration into a complete management framework.

1 Introduction

The World Wide Web has evolved as the largest network connecting users all around the globe and a most efficient resource of information and services. A Web service is a software component that is offered as a service on the Web through standard interfaces and communication protocols, and is a popular example of the current trend toward Service-Oriented Architecture (SOA) [11]. The potential that lies in dynamically composing multiple Web services offered by different organizations over the Internet into e-business processes can help achieve the goal of flexible online Business-to-Business (B2B) and Business-to-Consumer (B2C) relationships.

F. H. Zulkernine (✉)

School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada
e-mail: farhana@cs.queensu.ca

However, for the success of e-services, efficient management of Web services [6, 49] and Web services-based processes [15] is mandatory.

Systems management is traditionally defined as the administration of distributed systems [38] and involves functions such as fault management, configuration management, performance management, security, and accounting. The move to service-oriented architectures, and specifically to Web service-based applications in our case, is forcing a re-evaluation of this definition of management. Applications can now be defined at runtime through the composition of services, and this dynamic property of the workload means that services must be adaptable [15].

We can consider a simple example of a process to create a monthly sales report, which is illustrated in Fig. 1. The process retrieves summary data from two departmental databases and then builds a report. Each database is accessible through a Web service that allows data retrieval. In order to implement the monthly sales report process the client must identify the Web services providing the data, negotiate Service Level Agreements (SLAs) with each service, and compose and execute a workflow to produce the report. At the same time, each Web service must verify the ability of the client to retrieve the desired data, negotiate its SLA with the client, and then execute its part of the process while monitoring performance to ensure its SLA is satisfied. SLAs are contractual agreements between the service provider and the service consumer, which outline the expected Quality of Service (QoS), and are important to guarantee consumer satisfaction in business transactions.

The Web service management system in the above scenario needs to be able to support both the client and the Web services. In doing so, the system faces different challenges and is required to provide different kinds of support. We can, therefore, consider management from two perspectives. Server-side management, on the one hand, focuses on ensuring proper execution and QoS [1] by the service provider based on a set of prenegotiated SLAs [5]. Client-side management, on the other

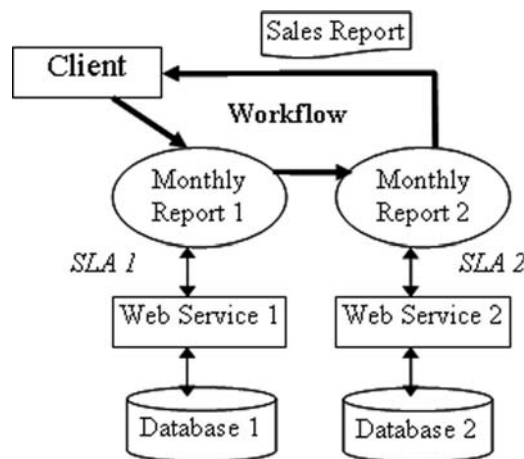


Fig. 1 Monthly sales report process

hand, focuses on ensuring QoS of the Web services-based process by supporting proper selection of services for the process [20], negotiation of SLAs [8] with each of the component services to meet the overall process QoS, definition, and execution of a workflow composed of the selected services [45], and finally monitoring the workflow to verify that the SLAs are satisfied [39].

The complexity of the above management tasks, as well as the complexity of the Web environment, make management of Web services and Web services-based processes a difficult problem and an interesting area of research. Autonomic Computing [18], which was introduced by IBM, can facilitate the management of large and complex systems by making them self-managing. This chapter considers the application of autonomic computing principles and techniques to both the server-side and client-side perspectives of Web service management. We present an example approach for each kind of management and examine how these approaches can be integrated to provide a complete management solution.

The remainder of the chapter is organized as follows. General background information on Web services and autonomic computing is provided in Section 2. Section 3 examines the related work. Section 4 describes the client-side and server-side perspectives of autonomic Web services management. It presents an example framework for each perspective from our own research and then discusses issues in integrating the two management perspectives. Our server-side Autonomic Web Service Environment (AWSE) framework [53] proposes an approach to developing autonomic Web services, whereas our client-side Comprehensive Service Management Middleware (CSMM) [50] proposes complete automation of client-side process management. Section 5 provides a summary and concludes the chapter.

2 Background

We first provide general discussions of Web services and autonomic computing and then outline the related work in autonomic systems and services management.

2.1 Web Services

Web services are software applications that are accessible on the Internet through standard communication protocols and interfaces. Features such as platform independence, Web accessibility, interoperability, ease of using Web services as wrappers for provisioning legacy applications, searching for, and composing services dynamically into business applications have greatly promoted the use of Web services. The Internet popularity has further leveraged the application of Web services in various aspects including remote resource management and as an efficient tool for outsourcing specific tasks.

Web service providers publish the information (Web access point or endpoint, functionality, parameters for the function calls, return values, protocols for binding,

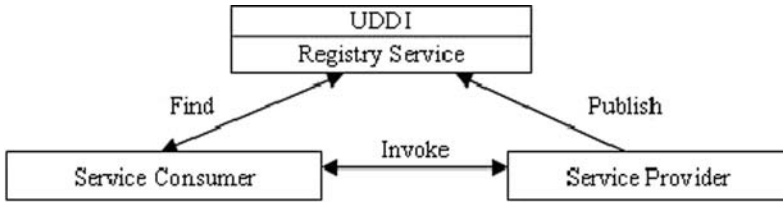


Fig. 2 Life cycle of a Web service

and communicating with the Web service) necessary to invoke a Web service as a WSDL (Web Service Description Language) [43] document. Service consumers look for required services in a registry called the UDDI (Universal Description, Discovery, and Integration) [28], which serves as the yellow pages for Web services. WSDL documents are linked to the UDDI. Once a desired service is found, a service consumer retrieves the WSDL document and invokes the Web service. Fig. 2 shows the lifecycle of Web services.

It is crucial to maintain interoperability among Web services published by different organizations to achieve the long cherished goal of B2B or B2C communication. The interoperability is achieved through strict implementation of numerous standard specifications in the implementation, publication and invocation of Web services. Extensible Markup Language (XML) [42] is used to write messages that are communicated with Web services using standard communication protocols. The most commonly used protocol for communication is SOAP (Simple Object Access Protocol) [41] messages over HTTP (Hyper Text Transport Protocol). SOAP messages are basically XML data bounded by headers and footers containing the messaging protocol, which is interpreted using a SOAP engine application. The SOAP engine translates the envelope of the message and after necessary preprocessing, passes the message to the appropriate Web service.

The use of standards allows multiple Web services from different organizations to be linked to each other to create business processes, which are called Composite Web Services. A Service composition created at design time is called a Static Composition, where appropriate services are only selected at design time but bound later at invocation time. In the case of Dynamic Composition, the services are selected, bound and invoked at the same time. Service composition provides an efficient way to create multi-organizational complex business processes. It greatly reduces the development and maintenance cost of traditional Enterprise Application Integration (EAI) [37] software while allowing task outsourcing, and using the most up-to-date service available at the time of service invocation. The example shown in Fig. 1 represents a composite Web service-based process, where the process of generating a sales report comprises two Web services that are invoked sequentially to provide a convenient and coherent on-demand Web-accessible software solution. This agility however, comes at the added cost and complexity of systems management both on the service consumer and the service provider's sides.

2.2 Autonomic Computing

Autonomic Computing [18] has emerged as a solution for dealing with the increasing complexity of managing and tuning computing environments. It is expected that Autonomic Computing will result in significant improvements in terms of system management and many initiatives have begun to incorporate autonomic capabilities into software components. Computing systems that feature the following four characteristics are referred to as Autonomic Systems:

- Self-configuring: Define themselves on the fly to adapt to a dynamically changing environment.
- Self-healing: Identify and fix the failed components without introducing apparent disruption.
- Self-optimizing: Achieve optimal performance by self-monitoring and self-tuning resources.
- Self-protecting: Protect themselves from attacks by managing user access, detecting intrusions and providing recovery capabilities.

Autonomic Computing will shift the responsibility for software management from the human administrator to the software system itself. The system should monitor its status and adapt itself accordingly to maintain a normal level of functionality at all times. The self-management is usually achieved by implementing a feedback control loop in the system which is called the MAPE [18] (Monitor, Analyze, Plan, and Execute) loop as shown in Fig. 3. The Monitor module collects the system's performance data; the Analyzer uses this data in light of the system policies and goals to determine whether or not the system is performing properly; the Planner determines if, and what action should be taken, and finally the Executor executes the suggested action to manage the system.

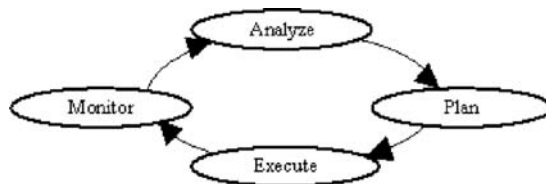


Fig. 3 The MAPE loop

3 Related Work

Autonomic systems address four self-management aspects. Most research on autonomic systems focuses on one or more of these aspects. The core part of an autonomic management system is a Controller, which is designed using either a performance feedback loop in a reactive manner, or a feed-forward loop in

predictive manner. Researchers propose different methodologies and frameworks to implement the controllers. Different search algorithms are proposed to search for the optimal values of the tunable configuration parameters. Various prediction logics are applied in different approaches in the case of predictive controllers. Typically, a controller can dynamically manipulate the control switches, modify configuration parameters, or implement additional queues based on the performance data to provide optimal throughput.

Cheng et al. [7] describes Autonomic Service Architecture (ASA) for automated management of both Internet services and their underlying network resources. They validate their framework by illustrating an autonomic bandwidth sharing scheme on a DiffServ/MPLS network that dynamically allocates necessary bandwidth to meet differentiated SLAs according to the monitored traffic load and policies specified for efficient resource utilization. ASA implements the Web Services Resource Framework (WSRF) [30] and uses a hierarchy of Autonomic Resource Brokers (analogous to IBM's autonomic element) to manage lower level services.

Liao et al. [22],[48] present a hierarchical agent infrastructure containing two types of agents for federated coordination of the agents to execute and manage Web service transactions. Task Agents encapsulate and control one or more Web services; multiple task agents form an Agent Federation; multiple agent federations can be grouped to form an Upper Agent Federation, and in each federation, multiple Management Agents perform service registration, negotiation, and transactions. A container of task agents implements the control logic to select and compose services from within a federation to provide a service required by the management agent.

Researchers at IBM [21] propose an architecture of a performance management system for cluster-based Web services. The system supports differentiated SLA provisioning, and performs dynamic resource allocation, load balancing, and server overload protection for multiple classes of Web services traffic. It applies a bi-level approach to management that implements a queuing algorithm at the inner level and a feedback control loop at the outer level to periodically adjust the scheduling weights and server allocations of the inner level. However, it requires users to subscribe to services before using them.

Chung and Hollingsworth [9] use a technique for resource sharing and distribution with Active Harmony servers, which dynamically reconfigure the roles of specific nodes in a cluster-based Web service environment to boost performance. They also implement an automated performance tuning infrastructure based on the workload, to enable adaptive tuning of a set of tunable parameters. The parameters are revealed by the applications and services through an Application Programming Interface. The core part of the server is a controller that implements optimization algorithms to determine the proper values of the tuning parameters.

Control theoretic approaches have been proposed by some researchers to implement the controllers. ControlWare is a middleware that embodies a control-theoretical methodology for QoS provisioning for software services [1]. The feedback control mechanism guarantees QoS through optimized allocation of system resources such as various queues and cache sizes to different processes.

Bennani and Menascé [2] create self-managing computer systems by incorporating mechanisms for self-adjusting the configuration parameters so that the QoS requirements of the system are constantly met. Their approach combines analytical performance models with combinatorial search techniques to develop controllers that run periodically to determine the best possible configuration for the system given its workload.

Birman et al. [3] extend the general architecture of Web service systems to add high availability, fault tolerance, and autonomous behavior. The architecture includes server and client side monitoring, a consistent and reliable messaging system using information replication, data dissemination mechanism using multicasting, and an event notification system.

A framework for deployment and subsequent self-configuration and self-healing of component-based distributed applications is proposed by Dearle et al. [14]. An Autonomic Deployment and Management Engine is used to find a configuration that satisfies the deployment goal, and the configuration is deployed automatically. If a deviation or change in the goal is detected, the configuration finder and deployment cycle is repeated automatically to generate a revised deployment.

Gurguis and Zeid [19] illustrate the self-healing aspect of autonomic Web services. To augment with autonomic features, they propose adding four different autonomic Web services, one for each of the four autonomic properties to be combined with the functional Web services to enable autonomic behavior. The MAPE loop is implemented by collaboration of four separate Web services where each service executes one of the four phases in the MAPE loop. Log files and Common Base Events are used for monitoring; a symptom database is used for diagnosing the problem, and a policy database is used to determine the proper recovery actions.

The self-configuration aspect is addressed by Reich et al. [36] through the implementation of an autonomic container based on the WSRF. They propose service migration as a remedy for solving overload or bottleneck problems based on a health status metric (H-metric) measurement. In a two level hierarchy, a service dispatcher at the upper level virtual container decides where a service should migrate to.

Pautasso et al. [33] describe a framework for the autonomic execution of Web service compositions using dynamic system configuration. A process control manager queues the requests for process execution, a navigator schedules the execution of processes from the queue, and a dispatcher makes the service calls for each process in the system. Based on the resource utilization and thresholds, the optimization policy determines the allocation of additional navigator and dispatcher threads dynamically during process execution.

Zeid and Gurguis [47] propose a high-level view of an autonomic Web services framework where each service is encapsulated within an autonomic resource shell that is managed by an autonomic manager. A collaboration manager manages access to multiple services and invokes them through a reputation manager. The framework does not provide a clear view of how autonomous properties are implemented.

The self-optimization aspect of autonomic management for Internet services is addressed by Bouchenak et al. [4] through dynamic resource provisioning strategy for varying workloads. The authors design and implement Jade, an environment to

provide autonomic management capabilities for distributed applications in general, which includes clustered multi-tier Internet services.

An Autonomic Web Services Net Traveler system is proposed by Monge and Martinez [25] where peer-to-peer (P2P) intelligent agents are used as brokers to coordinate, plan and perform Web service choreographies to render autonomic business processes. The authors lay out a preliminary layered architecture that focuses on scalability, fault tolerance, and availability aspects of autonomic process management.

Autonomic security mechanisms are usually addressed discretely from the other autonomic aspects. Some of the security systems can be added as a layer on top of an autonomic Web services system enabling its self-protection aspect. Dai et al. [13] propose an approach to detecting security problems using the feature recognition technique by virtual neurons, which are distributed in a compound P2P and hierarchical structure in the network. Park et al. [32] propose a policy-based Autonomic Protection System that applies Role-Based Access Control with an Intrusion Detection System, and allows self-adaptation of the security policies to suit various computing environments. Coetzee and Eloff [10] and Mecella et al. [24] both propose access control frameworks for Web services conversations pointing out the necessity to address the nature of repeated communication with Web services where one time access control may not be enough. The model demonstrated in [10] takes in account both trust and context awareness while [24] focuses on the importance of a tradeoff between the protection of the access control policies and the necessity to disclose partial policy information to the clients. Olson et al. [31] propose a third-party negotiation system for trust negotiation to gain access to a Web service.

3.1 Discussion

The system administrators typically have a specific set of goals for *QoS*, *Recovery*, *Security*, and *Resource Utilization* that they try to satisfy. Table 1 summarizes the various techniques typically used to address the different management goals.

Most of the research work on autonomic systems addresses one or more of the self-managing aspects, but not all. The most common techniques used to provide QoS are workload distribution using queues and priority features, resource sharing by scheduling algorithms, and dynamic parameter tuning using the MAPE loop [1, 4, 7, 9, 21]. Other approaches [3, 14, 19, 22, 25, 36] apply dynamic resource allocation that allows the reuse of the same resource for multiple purposes, service migration, automatic service deployment, replication, and configuration. These approaches are very specific to the system architecture and enable better recovery. However, these systems focus on only the availability property of the QoS, and not on the performance metrics.

Adaptive systems [14, 19, 22, 32, 36] that can adjust policies and decision making analogy dynamically with the systems' changing status are becoming increasingly popular. Agent-based systems are more autonomic due to the autonomous

Table 1 Management criteria and techniques

		Goals			
		QoS	Security	Recovery	Resource optimization
Techniques	• Workload identification and distribution	• Encryption	• Replication	• Resource sharing	
	• Workload adaptation	• Authorization	• Fault tolerance	• Resource distribution	
	• Dynamic resource allocation	• Access Control	• Adaptive tuning	• Queuing models	
	• Priority queue	• Intrusion detection	• Autonomic deployment		
	• Data multicast				
	• Notification				

nature of the agents, which is why they are used with Web services in many process management approaches [22, 25].

The WSRF framework uses Web services endpoints [19, 36] to tune parameters and manage resources which includes Web services. Many autonomic frameworks are utilizing the general accessibility of Web services to design frameworks composed of multiple levels of autonomic managers to manage Web service-based processes [22, 25, 33, 48]. However, the lower level management procedures are often not very well defined for those frameworks[33, 48].

The self-protection aspect of autonomic computing is often investigated under the paradigm of security, privacy and trust [10, 13, 24, 31, 32], and therefore, is not commonly addressed with the other management aspects. An integrated approach is deemed necessary to enable autonomic management of Web services and Web services-based processes. The basic management framework should be flexible enough to incorporate or function coherently with additional modules that provide autonomic protection or other useful features.

3.2 Research Trend

As systems are becoming more complex, different techniques are applied to design more effective autonomic systems keeping an eye on the four aspects of self-managing systems. Due to the versatility and the nature of accessibility of Web services-based systems, the primary management goals are QoS [47, 52] and security [10, 13, 31]. The most important QoS attributes are availability, reliability, guaranteed throughput, and response time [12]. While researchers are trying new algorithms, agent-based techniques, and methodologies to improve the performance of existing techniques to make systems more adaptive and reconfigurable, new techniques that use machine learning and other artificial intelligence methodologies are also being applied with the various standards of Web services such as the WSRF and the Web Services Distributed Management (WSDM) [20, 26, 49]. Management

of more complex systems is enabled using hierarchical management frameworks of collaborative management Web services [7, 53] and distributed management methodologies with a combination of the above techniques are used to manage Web services-based composite processes [15, 50].

4 Management Perspectives

We examine the issues involved in client-side and server-side management of Web services. We present our research on autonomic Web services management that addresses both client-side and server-side management. We highlight the principles and goals of our approaches, and present our ideology to integrate the two independent client-side and server-side management frameworks into a global solution to managing Web services and Web services-based processes.

4.1 Server-Side Management (AWSE)

On the service provider side, management of Web services places a difficult task on the system administrators due to the factors such as workload diversity and variability, increased emphasis on QoS, multi-tier architecture, service and network dependency, and advances in functionality, connectivity, and heterogeneity. Fig. 4 shows a number of components that are typically required to host a Web service. A client request for a Web service must pass through a number of layers of processing to reach the destination Web service. HTTP and Application servers are usual components of a Web server. The message processing software interprets the SOAP messages from the client and sends it to the proper service endpoint. The back end applications may include other third-party products, tools, hardware, software, and may involve the invocation of other Web services, which do the main processing and send the results back to the Web service interface application at the front end.

The AWSE illustrated in Fig. 5 is a framework for autonomic management of a Web services environment [40, 53]. In AWSE, system management and SLA

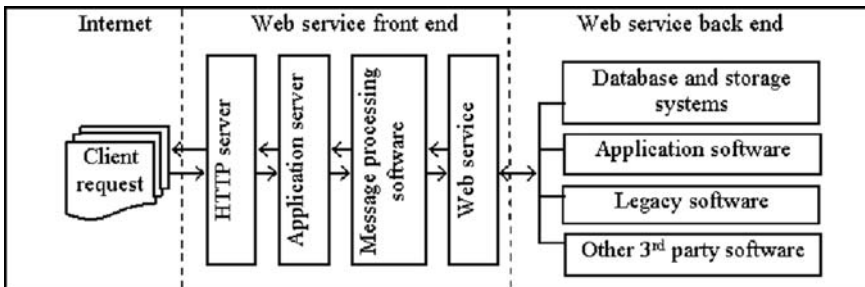


Fig. 4 Components of a Web service system

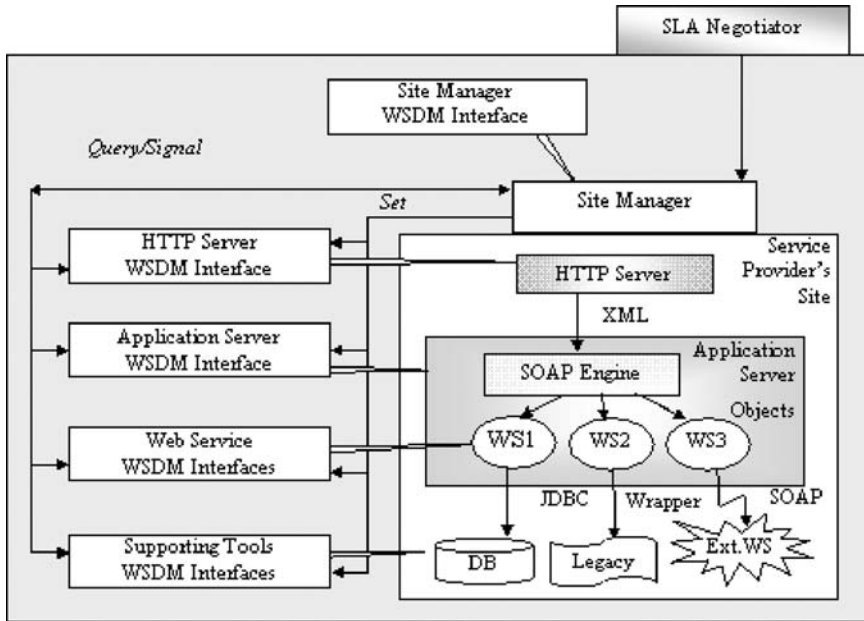


Fig. 5 AWSE architecture

compliance is facilitated by a hierarchy of autonomic managers. The higher level managers query lower level managers to acquire current and past performance statistics, consolidate the data from various sources, and use predefined policies and SLAs to assist in systemwide management. AWSE is based on the Autonomic Computing paradigm and employs widely accepted standards such as Web services and the WSDM [29] standard, thus allowing seamless integration of heterogeneous systems and standardized communication.

In AWSE, each of the individual components of a web hosting site, including the HTTP servers, application servers, database servers, and Web service applications, employ one or more associated autonomic managers. Each component is assumed to be autonomic, thus able to self-manage given component-specific performance goals. At the highest level of the hierarchy, a site manager, also an autonomic manager, monitors the overall performance of the site and provides service provisioning and management of the components, if necessary, to ensure overall system performance. The site manager also handles SLA negotiation between the site and potential clients.

The autonomic managers used in AWSE implement the standard MAPE loop using a reflective database-oriented framework [34]. A reflective system maintains a model of self-representation and changes to the self-representation are automatically reflected in the underlying system. In AWSE, the self-representation embodies the current configuration settings for the managed element, which control the performance of the element.

The rich capabilities of a Database Management System (DBMS) are used for data storage, creation of a knowledge base, and for controlled execution of the logic flow in the autonomic managers. The system knowledge base includes system topology, performance metrics, component-based and systemwide policies, and the expectations, or system goals. DBMS techniques such as triggers and stored procedures are used to implement the logic flow of the autonomic manager.

Although each individual component is self-managing, thus ensuring adequate performance of a component in isolation, this alone does not guarantee that the system as a whole is performing satisfactorily. In a distributed system such as a Web services environment, it is necessary for components to cooperate to attain system wide goals. In AWSE, the site manager is responsible for overall system management to ensure SLA compliance. It manages individual components by examining a component's current performance in light of the overall system performance. The site manager effects change on a component by adjusting its goal. The component will then self-configure to meet the new goal.

The site manager, therefore, must be able to communicate with individual components to retrieve performance metrics and to communicate revised goals. It also needs to be informed of any relevant events such as component configuration changes, or changes in performance that cannot be managed by the component itself. Thus communication between the site manager and the components of the Web services environment is critical.

In AWSE, management of components and communication of events is done through standard WSDM interfaces [29]. The main focus of WSDM is the manageable resource, which is a resource that exposes its manageability in a standard conformant way. In AWSE, a manageable resource is a component such as the DBMS, the HTTP server, or a Web service.

Management information for each manageable resource is accessible through a Web service endpoint called the Endpoint Reference (EPR). The EPR provides a location for the site manager, or other components to communicate with the manageable resource by means of SOAP messages. Manageable resources in WSDM are described using XML to specify the resource properties that support the manageability capabilities exposed by the managed resource. In AWSE, the self-representation of the system, that is, the current configuration settings for the managed resource, maps directly to the resource properties in WSDM. The component performance goal is also specified as a resource property. Resource properties can be retrieved and set by external components (namely the site manager in AWSE).

AWSE assumes that individual component performance data is exposed to other components, including the site manager. WSDM provides support for defining performance data using the concept of a Metrics capability. The Metrics capability supports metric information relevant to the performance and operation of the resource and allows the specification of metrics associated with each resource. These metrics can be retrieved by the site manager through the WSDM interface.

Using WSDM, the site manager can subscribe to receive event notifications when certain changes occur to a resource. Once subscribed, if the value of a property exposed by this capability changes, a notification is sent to the subscriber. This

mechanism allows the site manager to be kept informed of modifications in the system environment and, if necessary, to react accordingly.

4.2 Client-Side Management (CSMM)

On the consumer side, the process of composing multiple services to accomplish a certain task involves several steps as shown in Fig. 6. Based on the type of task, each of these steps can be quite complex. The dependency on the network and service providers support the fact that a good management system is mandatory to create and manage Web services-based workflows, particularly to realize the potential of creating and executing critical business processes.

The main steps that are involved in creating and executing a Web services-based workflow are listed below:

- Service Selection: Select a service based on some predefined criteria to complete a business process or replace a service to recover from failure.
- SLA Negotiation: Negotiate the SLAs based on service offerings and customer requirements of QoS.
- Workflow Orchestration and Execution: Design a workflow by organizing selected Web services with properly matched input and output parameters. Execute the workflow with proper checkpoints and exception handling procedures.
- Monitor and Error Report: Monitor the performances of each service in the workflow to verify compliance with the SLA, detect failure, and initiate quick recovery.

Client-side service management should facilitate all of the above and guarantee seamless execution of the workflow on the network.

We propose the CSMM framework as a flexible and versatile solution to client-side distributed management of autonomic Web services-based processes. It contains four main modules, and other accessory modules and repositories as shown in Fig. 7. Each of the four main modules is designed as an independent Web service and can be invoked separately to carry out one of the four major steps shown in Fig. 6. The workflow is deemed as autonomic because the framework implements a MAPE loop within itself which allows the workflow to be defined using an optimum set of available services and modified in case of failure. Furthermore, since all of the

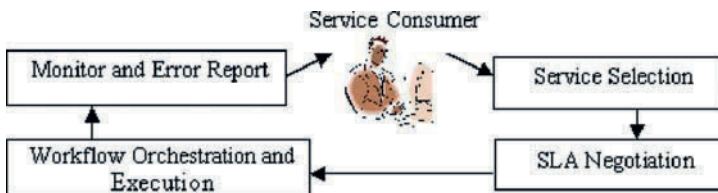


Fig. 6 Steps to execute a Web service process

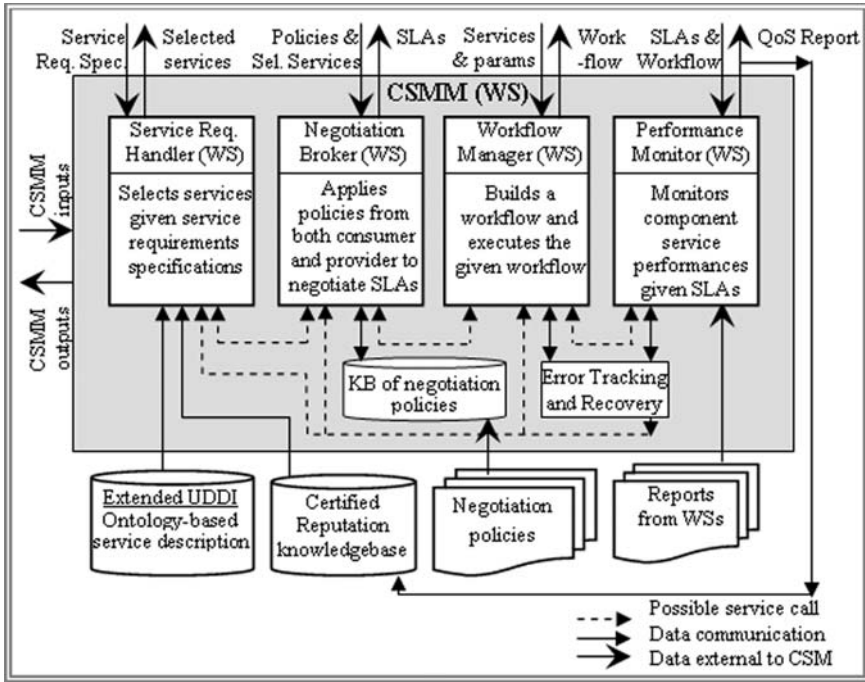


Fig. 7 Conceptual framework of the Comprehensive Service Management Middleware

four main modules provide Web service interfaces, they can be implemented based on the AWSE [40] framework, thus making each of the modules autonomic.

CSMM allows partial or complete automation of the job of defining and executing a Web services-based process. Consumers can use the four main modules independent of each other for the separate tasks of service selection, SLA negotiation, workflow orchestration and execution, and monitoring the workflow to ensure satisfaction of the SLAs. Using the comprehensive service of the CSMM requires clear specification of the process requirement, which is then interpreted and carried out sequentially by the different modules. If a failure is detected during the execution of the workflow, possible causes are investigated and plans are made accordingly to execute the proper recovery procedure. For Web services-based processes, a typical recovery mechanism would be to replace a failed service (unavailable or failed to meet the SLAs) with a similar service. To execute service replacement, notifications are sent to the appropriate modules first to find a service similar to the failed service and then to negotiate SLAs for the new service. Next, the original workflow is redefined with the replacement service and the workflow continues with no apparent anomaly from the consumer’s perspective.

Although automation of the above tasks of service selection, negotiation, orchestration, and monitoring are still popular topics of ongoing research, CSMM lays out a flexible general framework to combine different approaches to render a coherent

solution to autonomic Web services-based processes. The functionality of each module is described below in further detail.

4.2.1 Service Requirements Handler

Service Requirements Handler (SRH) is the first module in the CSMM. It finds required services for the consumer based on some specified selection criteria. It accepts specifications describing service requirements and returns a set of selected services in the order of execution. The extended UDDI and certified reputation knowledgebase are used to enable service discovery based on statistical data on service quality and reputation. The SRH draws from principles in the areas of automated service discovery [20, 44, 47] and ontology-based service description and selection [23].

4.2.2 Negotiation Broker

Once services are selected, an SLA is negotiated between the service provider and the service consumer based on consumer requirements and service offerings of the seller. The Negotiation Broker (NB) module takes an ordered list of selected services and the negotiation policies from all the service providers and the service consumers. The policies, which can be stored in a local repository for subsequent reuse, describe the context of the negotiators, their goals, the range of acceptable values for the issues, constraints, and preferences. This module performs automated negotiation locally as a trusted broker service and returns SLAs to both parties after a successful negotiation. The NB functionality is based on work from negotiation theory [35], machine learning [26], and automated negotiation [8, 16].

4.2.3 Workflow Manager

The Workflow Manager (WM) performs workflow orchestration, which is the design of a workflow for a business process by properly organizing selected Web services. It then executes the workflow, and ensures the application of proper corrective measures. The WM module takes an ordered list of selected services with necessary input parameters for each of them, and generates a formal specification such as the Web Service Business Process Execution Language [27] specification of a service orchestration. The WM builds on the large body of research on workflows and their formal representation as Petri Nets [46].

4.2.4 Performance Monitor

The Performance Manager (PM) monitors the performances of the component Web services in the workflow to verify compliance with the SLAs. The PM module takes the SLAs and workflow specification as input. If a service becomes unavailable or fails to meet the specified SLAs, it initiates necessary corrective actions through the Error Tracking and Recovery (ETR) submodule. The PM can also be used

to build a certified statistical QoS knowledge base from the performance reports collected for the different Web services in the workflow to enable reputation-based service selection. The PM builds on the research on message interception, formal representation of event-based systems as Petri Nets [45], and reputation-based service discovery [44].

4.3 Integration Issues

The AWSE and CSMM management frameworks discussed above address the two management perspectives and provide complementary management functionality. CSMM provides a very general and versatile solution that can accommodate a wide variety of approaches to automate the process management on the client-side. It ensures that overall process QoS goals are met by applying proper service selection approaches in the SRH module and accordingly conducting the SLA negotiation in the NB to satisfy the QoS goal of the total process. Failure in negotiation is reported back to the SRH, which then selects an alternative service, or modifies negotiation policies to reach an acceptable agreement. Since the resources for executing a Web service workflow are the services, the SRH also needs to select services carefully to make optimal use of resources. The execution of the workflow is monitored by both the WM and the PM. The WM implements proper checkpoints to monitor the progress of the process execution and reports any failure to the ETR. The PM monitors the SLAs of each component service within the workflow, analyzes the monitoring data against corresponding SLAs, and reports exceptions to the ETR. The ETR initiates proper recovery measures to provide a seamless workflow. CSMM provides comprehensive support for partial or complete automation of the tasks necessary for the execution of Web services-based processes over the Internet.

AWSE provides autonomic server-side management for the whole site that hosts Web services. The hierarchical autonomic elements in the AWSE framework allow collection and compilation of the status reports from each element through the WSDM interfaces, and provides a complete picture of the system status. Each autonomic element implements its own MAPE loop to adapt itself to the designated QoS model, which can be adapted by the upper level autonomic managers when necessary.

Integration of the AWSE and CSMM frameworks would provide a complete unified approach to all aspects of Web service management. The key advantage of the integration is that information from the two management perspectives can be fed into each other to provide more effective execution of the Web service-based processes. Monitoring data collected by AWSE for each service can be fed to the CSMM PM. The data can be used in the construction of the reputation knowledge base and by the NB in formulating more realistic service level objectives. The SLAs negotiated by the CSMM NB are used by AWSE to govern resource allocation and in turn control the performance of the Web service components. The CSMM could also interact with AWSE to obtain more detailed information in the case of execution errors or failures to meet SLAs.

We return to the monthly sales reporting scenario introduced earlier (see Fig. 1) to illustrate how AWSE and CSMM could interact with each other. A client initiates a request to CSMM that contains the process description, possible services to be selected, and the policies specifying preferences for SLA negotiation. The CSMM first invokes the SRH to select proper services. The SRH uses the UDDI and the reputation knowledge base to find services that best suit the requirements. The NB then negotiates the SLAs with the corresponding service providers. The AWSE Site Manager will, in this case, provide the NB with the server-side policies for SLA negotiation. Based on the policies, the NB tries to negotiate and come up with a set of SLAs that satisfies the policies specified by both parties. If the NB fails to reach a consensus, it replies back to the SRH to select an alternative service or if possible, modify the negotiation policies.

Once the department services are specified, the NB sends the SLAs to the corresponding service providers and invokes the WM. The WM links these services to create a workflow and contacts the PM to monitor the SLAs for the workflow before executing it. The PM communicates with the Site Managers of the two services and subscribes to be notified about the performance when the respective services are invoked. The WM executes the workflow and as the component services are invoked, the Site Manager sends performance reports to the PM. In the case of exception or violation of the SLAs, the ETR is notified and corrective actions can be initiated. Once the process is completed, the WM notifies the CSMM, which returns the monthly sales report to the client. Implementation details of the PM [52], the NB [51], and the server-side AWSE [53] are given in separate publications.

On the server-side, the Site Manager sets the performance goals for the different system components once the SLAs are received. The system then manages itself to meet the specified performance goals. When the PM subscribes to the Site Manager for performance reports, the Site Manager monitors the service request and sends a report of the performance data to the PM. The meta-data for relating the reports to specific processes are conveyed through the header section of the SOAP messages that are communicated for service invocations. The integration of AWSE and CSMM is facilitated by the use of standards and open source software in their development. As mentioned above, our prototype implementation of AWSE conforms to the WSDM standard which provides a common interface for the Web service components and managers. This interface can also be adopted for the various components of CSMM.

5 Summary

The two different perspectives of managing networked Web services-based processes, namely, the client-side and the server-side management, both put forth complex research challenges. Some of the primary concerns of the server-side management of Web services are the varying workload, vulnerability, and security threats of the Web media, and the complex multi-tier architecture and

heterogeneity of Web services environment. The client-side management includes service selection, SLA negotiation, orchestration and seamless execution of the Web service-based workflow with proper monitoring and recovery mechanisms in place to guarantee process QoS.

Autonomic management [18] is deemed to be an effective approach to managing complex and heterogeneous systems with fluctuating workload and can provide effective solutions to the problems of Web services management on both client and server-side. Researchers have proposed different approaches [17] and frameworks for both server-side and consumer-side autonomic management of Web services and services-based processes. We present the state-of-the-art research in the area of autonomic Web services management, most of which address only one side of the management and specific management aspects. Our research addresses both sides of Web services management using autonomic computing methodology. For client-side management, we present the CSMM framework that can facilitate automation of building and managing Web services-based processes. On the server-side, our AWSE framework provides a hierarchical layout of autonomic managers for managing the various components of a Web service environment. Finally, we illustrate an integrated approach to complete automation of Web services management using both the AWSE and the CSMM frameworks.

References

1. Abdelzaher, T., Stankovic, J., Lu, C., Zhang, R., and Lu, Y.: Feedback performance control in software services. In: *IEEE Control Systems Magazine*, Vol. 23(3), (2003)
2. Bennani, M., and Menascé, D.: Assessing the robustness of self-managing computer systems under highly variable workloads. In: *Proc. Intl. Conf. Autonomic Computing (ICAC'04)*, pp. 62–69, NY, USA (2004)
3. Birman, K., Renesse, R., and Vogels, W.: Adding high availability and autonomic behavior to web services. In: *Proc. Intl. Conf. Software Engineering (ICSE'04)*, pp. 17–26, Scotland, UK (2004)
4. Bouchenak, S, De Palma, N, Hagimont, D, Krakowiak, S., and Taton, C.: Autonomic management of internet services: Experience with self-optimization. In: *Proc. IEEE Intl. Conf. Autonomic Computing (ICAC'06)*, pp. 309–310, Dublin, Ireland (2006)
5. Cappiello, C., Comuzzi, M., and Plebani, P.: On automated generation of web service level agreements. In: *Proc. IEEE Intl. Conf. Advanced Information Systems Engineering (CAiSE'07)*, pp. 264–278, Trondheim, Norway (2007)
6. Casati, F., Shan, E., Dayal, U., and Shan, M. C.: Service-oriented computing: Business-oriented management of web services, *Communications of the ACM*, Vol. 46(10), (2003)
7. Cheng, Y., Farha, R., Kim, M.S., Leon-Garcia, A., and Won-Ki Hong, J.: A generic architecture for autonomic service and network management. *Computer Communications*, Vol. 29(18), 3691–3709 (2006)
8. Chhetri, M.B., Lin, J., Goh, S., Zhang, J.Y., Kowalczyk, R., and Yan, J.: A coordinated architecture for the agent-based service level agreement negotiation of web service composition. In: *Proc. Australian Software Engineering Conference (ASWEC'06)*, pp. 90–99, IEEE Computer Society, Washington, DC, USA (2006)
9. Chung I., and Hollingsworth, J. K.: Automated cluster-based web service performance tuning. In: *Proc. IEEE Conf. High Performance Distributed Computing (HPDC'04)*, pp. 36–44, IEEE, Honolulu, Hawaii (2004)

10. Coetzee, M., and Eloff, J.: A trust and context aware access control model for web services conversations. In: Proc. International Conference on Trust, Privacy and Security in Digital Business, (TrustBus'07), pp. 115–124, Regensburg, Germany, LNCS, Springer (2007)
11. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S.: Service-oriented computing: The next step in web services. *Communications of the ACM*, Vol. 46(10), 29–34, ACM, NY, USA (2003)
12. Dahlem, D., Nickel, L., Sacha, J., Biskupski, B., Dowling, J., and Meier, R.: Towards improving the Availability of service compositions. In: Proc. IEEE Intl. Conf. Digital Ecosystems and Technologies (DEST'07), pp. 67–70, IEEE, Cairns, Australia (2007)
13. Dai, Y., Hinchey, M., Qi, M., and Zou, X.: Autonomic security and self-protection based on feature-recognition with virtual neurons. In: Proc. IEEE Int. Symposium of Dependable, Autonomic and Secure Computing (DASC'06), pp. 227–234, Washington, DC, USA (2006)
14. Dearle, A., Kirby, G., and McCarthy, A.: A framework for constraint-based deployment and autonomic management of distributed applications. In: Proc. Intl. Conf. Autonomic Computing (ICAC'04), pp. 300–301, NY, USA (2004)
15. Dustdar, S.: Towards autonomic processes and services. In: Proc. Intl. Working Conf. Business Process and Services Computing (BPSC), pp. 13–19, Leipzig, Germany (2007)
16. Faratin P., Sierra, C., and Jennings, N.: Negotiation decision functions for autonomous agents. *Intl. Journal of Robotics and Autonomous Systems*, Vol. 24(3–4), 159–182 (1998)
17. Farrell, J.A., and Kreger, H.: Web services management approaches. *IBM Systems Journal*, Vol. 41(2), 212–227 (2002)
18. Ganek, A., and Corbi, T.: The dawning of the autonomic computing era. *IBM System Journal*, Vol. 42 (1), 5–18 (2003)
19. Gurguis, S., and Zeid, A.: Towards autonomic web services: achieving self-healing using web services. In: SIGSOFT Software Eng. Notes, Vol. 30(4), 1–5 (2005)
20. Jakob, M., Healing, A., and Saffre F.: Mercury: Multi-agent adaptive service selection based on non-functional attributes. In: Proc. Intl. Workshop Engineering Emergence in Decentralised Autonomic Systems (EEDAS'07), Jacksonville, FL, USA (2007)
21. Levy, R., Nagarajarao, J., Pacifici, G., Spreitzer, M., Tantawi, A., and Youssef, A.: Performance management for cluster based web services. In: IBM Technical Report (2003)
22. Liao, B., Gao, J., Hu, J., and Chen, J.: A federated multi-agent system: Autonomic control of web services. In: Proc. Int. Conf. Machine Learning Cybernetics (ICMLC'04), Vol. 1, pp. 1–6, IEEE, Shanghai, China (2004)
23. Maximilien, E., and Singh, M.: A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, Vol. 8(5), pp. 84–93 (2004)
24. Mecella, M., Ouzzani, M., Paci, F., and Bertino, E.: Access control enforcement for conversation-based web services. In: Proc. Int. Conf. World Wide Web (WWW '06), pp. 257–266, Edinburgh, Scotland, ACM, New York (2006)
25. Monge, H., and Martinez, T.: AWS-Net Traveler: Autonomic web services framework for autonomic business processes. In: Proc. IEEE Int. Conf. Services Computing (SCC'05), Vol. 2, pp. 270–272, Orlando, FL, USA (2005)
26. Narayanan, V., and Jennings, N.: Learning to negotiate optimally in non-stationary environments. In: Proc. Intl. Workshop Cooperative Information Agents (CIA'06), pp. 288–300, Edinburgh, UK, ACM (2006)
27. OASIS: WS-BPEL (Web Services Business Process Execution Language) 2.0 Draft. At: <http://www.oasis-open.org/committees/download.php/14616/wsbpel-specification-draft.htm> (2006)
28. OASIS: UDDI Technical Committee Specification, v 3.0.2. At: <http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm> (2005)
29. OASIS: WSDM (Web Services Distributed Management): Management Using Web Services (MUWS 1.0), Part 1 (2005)
30. OASIS: WSRF (Web Services Resource Framework), v 1.2 (2006)

31. Olson, L., Winslett, M., Tonti, G., Seeley, N., Uszok, A., and Bradshaw, J.: Trust negotiation as an authorization service for web services. In: ICDE'06 Workshops, Vol. 21, IEEE, Atlanta, GA, USA (2006)
32. Park, S., Kim, W., and Kim, D.: Autonomic protection system using adaptive security policy. In: Proc. Int. Conf. Computational Science and Its Applications – ICCSA'04, LNCS 3045, pp. 896–905, A. Laganà et al. (Eds.), Assisi, Italy (2004)
33. Pautasso, C., Heinis, T., and Alonso, G.: Autonomic execution of web service compositions. In: Proc. Int. Conf. Web Services (ICWS'05), Orlando, FL, USA (2005)
34. Powley, W., and Martin, P.: A reflective database-oriented framework for autonomic managers. In: Proc. Int. Conf. Autonomic Systems (ICAS'06), pp. 57–62, San Jose, CA, USA (2006)
35. Raiffa H.: The Art and Science of Negotiation. Harvard University Press, Cambridge, USA (1982)
36. Reich, C., Banholzer, M., Buyya, R., and Bubendorfer K.: Engineering an autonomic container for WSRF-based web services. In: Proc. Int. Conf. Advanced Computing Commun. (ADCOM'07), pp. 277–282, Guwahati, India (2007)
37. Seth, M.: Web Services – A Fit for EAI. White Paper, At: <http://www.developer.com/tech/article.php/1489501> (2002)
38. Sloman, M.: Policy-driven management for distributed systems. *Journal of Network and Systems Management*, Vol. 2(4), pp. 333–360 (1994)
39. Ta, X., and Mao G.: Online end-to-end quality of service monitoring for service level agreement verification. In: Proc. IEEE Int. Conf. Networks (ICON'06), Vol. 2, pp. 1–6, Singapore (2006)
40. Tian, W., Zulkernine, F., Zebedee, J., Powley, W., and Martin, P.: An architecture for an autonomic web services environment. In: Proc. Joint Workshop Web Services Model-Driven Enterprise Information Syst. WSMDEIS (ICEIS'05), Miami, FL, USA (2005)
41. W3C: SOAP (Simple Object Access Protocol) Part 1: Messaging Framework, v 1.2. At: <http://www.w3.org/TR/soap12-part1/> (2003)
42. W3C: XML (eXtensible Markup Language). At: <http://www.w3.org/XML/> (2004)
43. W3C: WSDL (Web Services Description Language), v 2.0 (Working Draft). At: <http://www.w3.org/2002/ws/desc/> (2005)
44. Xu, Z., Martin, P., Powley, W., and Zulkernine, F.: Reputation-enhanced QoS-based web services discovery. In: Proc. IEEE Intl. Conf. Web Services (ICWS'07), pp. 249–256, Salt Lake City, Utah, USA (2007)
45. Yang, Y., Tan, Q., Xiao, Y., Yu, J., and Liu, F.: Exploiting hierarchical CP-nets to increase the reliability of web services workflow. In: Proc. International Symposium Appl. Internet (SAINT'06), pp. 116–122, IEEE Computer Society, Washington, DC, USA (2006)
46. Yu, T., Zhang, Y., and Lin, K.: Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Transactions on the Web*, Vol. 1(1), Art. 6, pp. 1–26 (2007)
47. Zeid, A., and Gurguis, S.: Towards autonomic web services. In: Proc. ACS/IEEE Int. Conf. Computer Syst. Appl. (ICCSA'05), pp. 69–73, IEEE, Cairo, Egypt (2005)
48. Zhang, F., Ji, G., Guo, H., Zhu, P., and Liao, B.: Autonomic management of web services based on federated multi-agent system. In: Proc. World Congress Intelligent Control and Automation, Vol. 2, pp. 6949–6953, Dalian, China (2006)
49. Zulkernine, F., and Martin, P.: Web services management: Towards efficient web data access. In: *Web Data Management Practices: Emerging Techniques and Technologies*, pp. 266–288, A. Vakali, and G. Pallis (Eds.), Idea Group of Publishing, PA, USA (2006)
50. Zulkernine, F., and Martin, P.: Conceptual framework for a comprehensive service management middleware (best paper award). In: Proc. Int. IEEE Workshop Service Oriented Architectures in Converging Networked Environments (SOCNE'07) with AINA'07, Niagara Falls, Canada (2007)
51. Zulkernine, F., Martin, P., Craddock, C., and Wilson, K.: A policy-based middleware for web services SLA negotiation. Accepted in: Proc. IEEE International Conference Web Services (ICWS'08), IEEE, Beijing, China (2008)

52. Zulkernine, F., Martin, P., and Wilson, K.: A middleware solution to monitoring composite web services-based processes. Accepted in: Proc. of the IEEE Congress on Services (SERVICES'08) Part II, Workshop on Service Intelligence and Computing (SIC) at IEEE International Conference on Web Services (ICWS'08), IEEE, Beijing, China (2008)
53. Zulkernine, F., Tian, W., Powley, W., Martin, P., Xu, T., and Zebedee, J.: Autonomic web services environment using a reflective database-oriented approach. *Journal of Ubiquitous Computing and Communication Special Issue on Autonomic Computing and Communications* (2008)

Concepts for Self-Protection

Tanja Zseby, Heiko Pfeffer, and Stephan Steglich

Abstract Network protection should be a number one priority on every network operator's list. Even the best network is useless, if an intruder can gain control. Although the research community has been working in this field for decades, we are still at a far remove from networks where successful attacks are the exception. Scant deployment of security solutions is not the only reason. The fast evolution of protocols and applications and the permanent emergence of new attacks build an extremely dynamic environment in which protection becomes a tough challenge. Classical attack prevention techniques are not sufficient to deal with new and unexpected incidents. The immense administrative burden on users and providers calls for automation of security tasks and protection features as an integral part of future networks. However, network self-protection requires permanent awareness and the flexibility to re-act. Sophisticated observation and analysis techniques, cooperation, and information sharing together with learning concepts are crucial to achieve this goal. Autonomic communication provides a framework in which self-protection concepts can be developed.

1 Introduction

An increasing number of critical businesses and communities rely on the Internet. Many require and assume an uninterrupted and secure operation of networks. Even so, ever since the advent of the communication networks, attackers have tried to disturb, interrupt and destroy network operation. The quantity and quality of attacks is increasing. They have rapidly evolved from playful but not harmless competitions in hacker communities to instruments of warfare and organized crime. The upshot is immense financial loss, compromised data, and unacceptable constraints on users who try to set up more or less secure environments.

Internet research in all areas should be strongly coupled with concerns about security threats. The best, fastest, and most efficient network is useless, if an attacker

T. Zseby (✉)
Fraunhofer Institute Fokus, Berlin, Germany
e-mail: tanja.zseby@fokus.fraunhofer.de

can gain control. The rule stipulating that security considerations must be added to each new Internet standard given in the Internet Engineering Task Force (IETF), is only one step in the right direction.

With the increasing heterogeneity of the current Internet at all layers, its enduring growth and the dynamic nature of Internet topologies and traffic patterns, classical methods of attack prevention (authentication, secure protocols, firewalls, etc.) are no longer sufficient to cope with upcoming threats. Attackers are far away from using the full potential of current Internet vulnerabilities. And each new technology usually brings its own new vulnerabilities. We have to find answers soon before the wake-up call of a devastating super attack that wipes out the basis of Internet operation.

Autonomic communication provides a framework for incorporating more functionality into network nodes to introduce self-management and self-protection capabilities. As a supplement to classical methods of attack prevention, autonomic communication includes concepts of attack detection and defense based on node cooperation, adaptation, and learning. With the concept of learning from errors, Internet security is strengthened by each new assault.

2 Network Security

Awareness about the importance of network security has increased. Nevertheless, incident prevention activities still remain fragmentary. The Internet is a patchwork of differently administered networks. Security policies and their enforcement, if applied at all, vary immensely.

Pure network operation is the predominant goal. With the increasing complexity of the Internet, this would pose a challenge even in a world without adversaries. Securing the network requires additional effort and costs. And, crucially, it also requires permanent awareness. New protocols and applications continuously introduce new vulnerabilities. Lack of experience with new features, misconfiguration, and undetected vulnerabilities in new software lead to the constant emergence of new attack targets. Attempts to secure the Internet are forever competing with new challenges by highly motivated and increasingly organized and skilled attackers.

In computer networks we can distinguish the following security objectives:

- *Access control*: Protection of network resources against unauthorized access.
- *Confidentiality*: Protection of data against unauthorized disclosure.
- *Integrity*: Protection of data against unauthorized modification or discarding.
- *Availability*: Protection of access to and usability of network resources for authorized entities.
- *Non-Repudiation*: Protection against false denial of actions that a user or entity has performed.

In many networks servers for Authentication, Authorization, and Accounting (AAA) are used to control access. Firewalls, intrusion, and virus detection have

become standard in networks and end systems. New protocols and methods have been standardized that could increase security in the Internet. Even so, they are only sparsely deployed. Even if the awareness is there, many providers are bothered about the additional administrative effort and the learning curve that the deployment of new solutions involves. It takes too long for early adopters to pave the path for the majority.

Furthermore, high dynamics and complexity mean that user education will remain incomplete. The willingness of users to give up any convenience in network usage (e.g., opening arbitrary ports to enable gaming) for a diffuse security goal is limited. The need for user participation in security related activities (e.g., confirming security updates) should be minimized to prevent protection activities leading to comfort reduction. For defending a network against attacks we distinguish four phases: *Prevention*, *Detection*, *Defense*, and *Forensics* as illustrated in Fig. 1.

The ideal case is *prevention* of attacks. Prevention comprises all methods applied in order to avoid attacks beforehand. This means that methods are used to control access to devices and network resources and to ensure data confidentiality and integrity. This includes authorization and authentication techniques, trust establishment as well as encryption and filtering of traffic (firewalls). Prevention is only possible for attacks that are known or can be predicted.

If prevention fails, *detection* is the next best possibility for dealing with an incident. Detection is the process of discovering attacks, attack preparation, or any other malicious activities. This is usually done by data analysis. If attack or preparation activities are detected, actions should be invoked for *defense*. After the attack it is often useful to analyze what happened before and during the attack. Post-attack analysis of traffic is called *forensics*. This is done in order to learn how detection and defense methods can be improved for future incidents.

All this increases administrative costs for providers and enlarges the complex restrictive rule sets that users have to comply to. In all this the desire to reduce human intervention is evident. Security has to become an integral part in future networks. Technology can help to automate processes. The high dynamics in network evolution and in attack attempts call for the incorporation of learning techniques. Besides the observation and analysis of current and past events that forms the node's own experience, it can also profit from the experience of others. Cooperation and information sharing is the key. As we shall describe later, this is challenging at many levels.

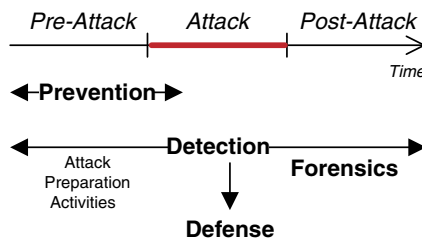


Fig. 1 Four phases of defense

3 Autonomic Communication

Autonomic communication describes an approach to automate decision processes in communication networks. The goal is to move decision-making processes toward network elements in order to minimize human intervention for network management and protection. Thus networks should be able to manage and protect themselves, based only on high-level goals set by users and operators.

Decision cycles have been used and explored in many disciplines. IBM started to apply decision cycles to computer systems under a paradigm called “autonomic computing.” The goal was to make systems self-managed in the sense that human intervention is minimized. Users only define high-level business rules and systems configure and optimize themselves to fulfill these goals. IBM distinguishes five levels that a system can reach: basic, managed, predictive, adaptive, and autonomic. The basic level is the lowest level and stands for systems in which all elements are managed by a human operator. Autonomic describes the most advanced level. Autonomic Systems are dynamically managed only by high-level business rules and policies.

In order to achieve this, IBM introduced the concept of autonomic managers. Autonomic managers implement an intelligent control loop that consists of four function blocks: Monitor, Analyze, Plan, and Execute (MAPE). Monitoring functions collect and aggregate information. Analysis functions correlate data and model situations to provide the basis for prediction. Planning provides functions to plan actions in accordance to defined policies in order to achieve specific goals. Execution functions initiate and control the execution of the plan [38]. In [48] Brent Miller defines several self-management attributes that an autonomic computing architecture should provide in order to automate the work of administrators: self-configure, self-heal, self-optimize, and self-protect, also known as the self-CHOP concept. Self-protecting is defined as the ability of a system to “anticipate, detect, identify, and protect against threats” [38]. This includes detection of hostile behavior and the performance of counteractions.

Autonomic communication can be seen as autonomic computing applied and adjusted to communication systems. In [18] the authors present an autonomic control loop that consists of function blocks to Collect, Analyze, Decide, and Act, along the lines of the MAPE concept in autonomic computing. If applied to computer networks the decision cycle has to be performed in and among network nodes. The main goal is to enable network nodes themselves to reach decisions about traffic handling or node configuration by collecting and interpreting the required information. Routing can be seen as an early example of autonomic communications. Routers share information and themselves make the forwarding decisions based on the analysis of the collected information.

A central point is the establishment of situation awareness. Situation awareness is the result of the observation and orientation steps and provides the basis for making sound decisions. It describes the state that evolves from constant observation and analysis of the environment. In the observation phase critical factors in the environment are perceived. With limited resources, it is important to concentrate on the most

relevant factors. This is not a trivial matter. The factors essential for making a good decision are not necessarily known a priori. Especially, in terms of new unknown situations, it is often unclear which factors have an impact. Storage of unprocessed observations is often required for learning. In some cases it is possible to analyze a posteriori which factors should have been observed and analyzed in order to make a better decision.

The orientation phase includes inference for understanding what the factors mean with regard to the decision maker's goals. It should also include prediction methods in order to prognosticate what might happen in the near future. This is required to see how the situation evolves and to detect critical developments in advance. Furthermore, prediction techniques help to assess how different alternative decisions would influence the situation.

A prerequisite for autonomic communication is improved situation awareness inside network nodes. Nodes need to collect relevant information in order to form a basis for local decisions. They need to share local data and decisions with other nodes for cooperative decision making. The observation of the behavior of neighbor nodes can also assist in assessing their integrity (e.g., whether they behave in accordance to their assigned roles, whether they origin or transfer malicious traffic, whether they are overloaded). This information helps to weight information from neighbors and prevent them from following wrong suggestions (e.g. originated by malfunctioning or infected nodes).

4 Generic Principles for Network Self-Protection

Following Sect. 3 we may define the following building blocks for enabling self-protection capabilities in network nodes:

- Prevention
- Establishment of Situation Awareness (Monitor and Analyze)
- Decisions Making (Plan)
- Re-configuration (Execute)

In the following we further elaborate existing techniques and challenges for these building blocks.

4.1 Prevention Strategies

Prevention mechanisms can be considered as a network's or system's first line of defense. The first step is prevention by design, i.e., designing systems, software, and protocols in a way that makes them difficult to attack. The second step is to prevent attacks during operation. This is typically achieved by controlling access to resources and information. However, the main drawback of prevention techniques is

their inability to cope with unknown and unpredictable attacks; they fail with new, so-called “zero-day attacks.”

The general idea of prevention by design is similar to that of model checking approaches for system verification [14]. Model checking procedures provide a mathematical proof for whether a system, given as a model \mathcal{M} , meets a certain specification *spec*. Prevention techniques borrow from the same idea. Assume that *spec* holds a description of an action that involves both the local node and a remote one. This action may be the exchange of a message, the transfer of a document or a part of application logic, a packet forwarding or an access grant for a specific part of local memory. Here, prevention techniques have to evaluate whether the specification *spec* can be applied to the system \mathcal{M} without violating its specific security requirements.

During operation, useful techniques to prevent attacks on the network are establishment of trust among communication partners, discarding of unwanted or suspicious traffic (firewalls), or techniques for data encryption, authentication, and authorization of users and network nodes. Sometimes the hiding of information about the network structure (e.g., by using network address translation) is useful to limit attack potential.

Within the networking area a variety of prevention techniques is known (e.g., secure protocols, access control) and partly deployed. Thus we shall here concentrate on additional techniques that can be applied at service level to improve the overall prevention. Such techniques are especially required in ad hoc environments, where nodes can leave and join at any time. Below we present prevention mechanisms based on service composition models that are suitable to improve self-protection in ad hoc and hybrid environments. We also discuss their possible integration in lightweight mobile middleware.

4.1.1 Hybrid Networks

In the terms of service level, prevention mechanism are of prime importance within ad hoc and mobile environments where nodes are typically not connected by a fixed infrastructure. They enter or leave each others connection range dynamically and spontaneously establish ephemeral connections.

Often such spontaneous connections between devices often cannot be established in a direct and secure way, e.g., through authentication procedures. This means they have to prevent undesirable behavior by estimating the other nodes’ trustworthiness and reputation. Networks featuring both fixed (infrastructure-based) and mobile nodes are referred to as “*hybrid networks*.” Mobile nodes imply a change in the network’s topology through their movement. They establish new connections based on their current geographical location and their connection range. Mobile nodes are often restricted in their computing capability due to size and power constraints. Nodes with severe restrictions, like sensor nodes or small user devices such as cell phones or smartphones, are called “tiny nodes.”

A group of nodes where each node is connected by single-hop or multi-hop with every other node is referred to as an *island* of nodes. Within these islands

services can collaborate with one another to fulfill a request from a user or another service. Such joint processing and execution of service compositions has been surveyed various times [10, 24, 56]. Within the following, we will investigate the possible integration of prevention mechanism into such collaborative schemes of services.

4.1.2 Prevention Mechanisms

We distinguish the self-protection mechanisms of nodes within hybrid networks by two dimensions. First, we consider at which *point in time* the mechanisms is integrated within a service; second, we focus on the *class of undesired behavior* to be prevented. For the first dimension, we estimate the prevention mechanism's point of integration in the service life cycle. The various phases of a traditional service life cycle are depicted in Fig. 2.

The first part of the life cycle, referred to as the designtime of a service, deals with a service's model, implementation, and verification. Designtime is generally the part of a service life cycle covered by software engineering. Here, prevention procedures are integrated during the designtime of a service, and are accordingly expressed as fixed design patterns for detecting attacks and malicious system behavior. On the other hand, the runtime phase of a service covers the service's behavior during its execution. Some prevention mechanisms do not only rely on predefined prevention mechanisms, but enable improvement of corresponding decision-making procedure through adaptation during service runtime. A prime example of this class of approaches are learning based prevention mechanisms that base their estimation

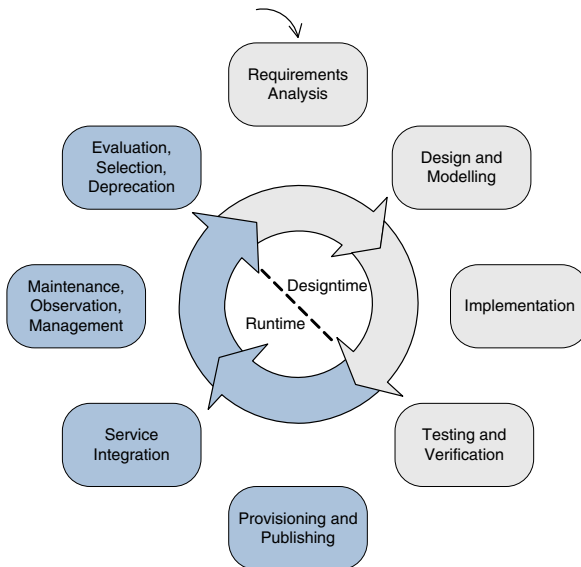


Fig. 2 A traditional service life cycle divided into designtime and runtime [52]

of a remote node’s trustworthiness on a trust value that is updated after special events or transactions between nodes.

The latter dimension, i.e., the class of prevented behavior, separates malicious from selfish behavior. Here, a behavior that aims at attacking other nodes is considered as malicious, while selfish behavior of a node entails security or performance problems for other nodes. Note that selfish behavior is not necessarily malicious, neither is every malicious behavior provoked by a selfishly acting node.

We consider four general classes of prevention mechanisms for hybrid networks:

- Decision Rules
- Virtual Currencies
- Reputation Evaluation
- Novel Service Life Cycles

Based on the two dimensions introduced above, approaches can be classified as in Fig. 3.

Decision Rules—Decision Rules are defined by the system developer in order to restrict service behavior and interaction based on a fix set of rules. Buford et al. [8] coined the term *composition trust bindings* by specifying a set of rules that define the composability of various services from different sources. Here, the mechanism can be used to secure both the path for service invocation and content handling. The rules are hard-coded during designtime and thus static. Since the allowed bindings are established by users based on their confidence in

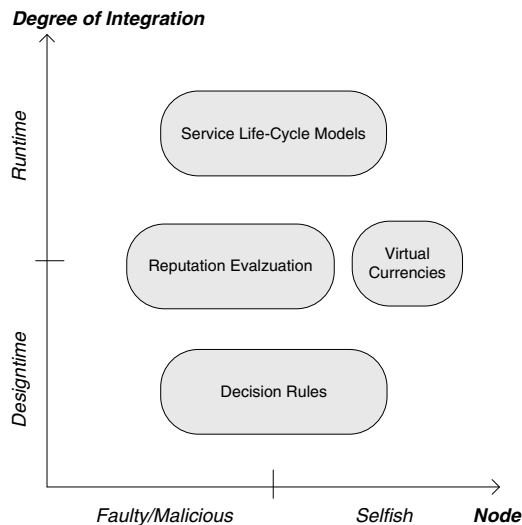


Fig. 3 Two-dimensional categorization of prevention mechanisms

specific services, the approach is suitable for the prevention of malicious as well as selfish nodes.

Virtual Currencies—The principle of virtual currencies is based on the observation that a service execution for a remote node is associated with specific costs such as resource and energy consumption or “real” money. Thus actions performed on remote nodes have to be paid for with virtual money, i.e., a specific amount of virtual money is retracted from the service requester and credited to the service provider. In order to pre-empt the selfish behavior of nodes, each node is assigned a limited amount of a virtual currency, forestalling that nodes act mainly as service requesters without providing any value to the community. Key examples of virtual currency schemes are Sprite [67] and Nuglets [9].

Reputation Evaluation—Reputation schemes allocate a degree of reputation to each participant in a given community. In commerce, this principle has been successfully implemented in online stores such as Amazon or eBay where each user owns a reputation based on ratings from other users after a completed transaction. Within ad hoc environments, nodes can be equipped with reputation values in order to identify especially uncooperative or selfish nodes. Examples of reputation evaluation approaches are Confidant [7] and Ocean [4].

Novel Service Life Cycles—While many prevention techniques rely on the execution of fixed rules in order to process incoming requests for service provisioning, approaches inspired by artificial intelligence or social science such as reputation evaluation or virtual currencies advance those principles from designtime to runtime. This development mainly relies on learning algorithms enabling adaptation of node behavior during service execution. To push prevention mechanisms further toward the runtime of a service, novel service life cycles are proposed that aim at integrating the least possible amount of complexity during the designtime of a service and that rely on advanced adaptation techniques during services’ runtime. One example of such life-cycles is the bio-inspired service life-cycle proposed by Pfeffer et al. [44, 52], where the biological concept of evolution has been used to model emerging behavior. This advanced short-term service adaption to long-term service evolution. The abstract of the bio-inspired service life cycle is given in Fig. 4.

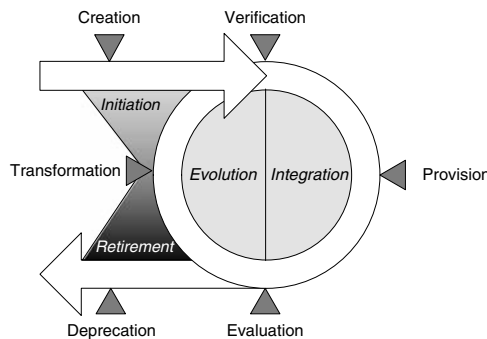


Fig. 4 A bio-inspired service life cycle [44]

The key feature of this novel service life cycle is the evolution phase that follows service creation and integration. Service evolution is realized in two parts. First, service compositions are evolved by the application of genetic operators, i.e., their structure is reorganized by either mutating a single service composition or combining two existing service compositions through a so-called crossover. The results on this transformation of service compositions through genetic operators were published by Linner et al. [45]. Transformation of a finite set of service compositions leads to a great amount of new service compositions. Therefore, the second part of the evolution phase comprises a selection mechanism that selects the fittest service compositions for further reproduction while discarding the others. This selection is based on a fitness function, that can be designed in accordance with the system to be evolved. In the context of self-protecting services, a fitness function can be defined that favors services with a good reputation, and eliminates malicious or selfish services over the long term. Thus, the whole system evolves toward a system of trustworthy nodes.

4.1.3 Realizing Trustworthy Service Compositions

In order to realize self-protecting systems, various approaches have been proposed, ranging from classic Autonomic Computing (AC) approaches driven by industries such as IBM [40, 60, 61] to bio-inspired organic middlewares introduced by Pietzowski et al. [53–55]. In the following section, we will outline a middleware for collaborating nodes within ad hoc environments as proposed by Jacob et al. [39]. The Middleware is based on *SmartWare* [43], a lightweight middleware enabling autonomic communication via REST [32].

SmartWare decouples autonomic behavior from actual services by featuring three different kinds of containers. First, a Service container holds the services available on the according device. Secondly, the Mediator container holds a set of mediators encapsulating lower level autonomic behavior such as service discovery. Thirdly, the Interaction container encompasses a set of possible interaction mediums such as the semantic data space [46], enabling a loosely coupled interaction of services and mediators. By outsourcing the autonomic capabilities of services into mediators, services become more lightweight and thus more likely to be executable on tiny nodes as introduced in Sect. 4.1.1. Moreover, not all mediators have to be executed on every node. In case a device is not capable or willing to host a specific mediator, the according functionality can be proxied by remote mediators on other devices.

The middleware for mobile collaborating devices proposed by Jacob et al. [39] builds on top of *SmartWare* and provides higher-level functionality, enabling collaborative behavior of multiple services. Thus the middleware provides a joint state management and eventing system and bases on principles of autonomic communication such as interaction patterns enabling loose coupling and dynamic service discovery. Prevention mechanisms are implemented by a *Trust Mediator* with a learning algorithm to update the trust values held for each device/service pair on special occasions, e.g., after a transaction with a service, a change in the environmental context, or a variation of network topology.

4.2 Establishment of Situation Awareness

If prevention fails, the next step is detection of attack preparation or already ongoing attack activities in the network. A pre-requisite for this is establishment of *situation awareness*. This is typically done by observing and analyzing network node behavior and traffic flows. The situational view provides the basis on which to make a decision about the current threat level. From this adequate countermeasures can be activated.

Establishing situation awareness is not a simple task. It can be subdivided into three levels:

- *Perception*: Observe the relevant factors in the environment.
- *Inference*: Understand what those factors mean for the decision maker's goal.
- *Prediction*: Make a prognosis of what could happen next.

Network traffic is usually extremely dynamic and difficult to predict. This means that the view of the situation needs to be constantly updated. The ideal would be to gain a complete picture of the network. That would involve observing every packet at each network node with deep packet inspection to read each packet's content. With this we could detect even sophisticated application-level attacks at least a posteriori. But observing all packets at all nodes is impossible. We cannot measure everything everywhere.

First of all we have to deal with *resource limitations*. Processing power, storage, and transmission capacity are limited. Network measurements are only support functions for network operation and should not influence network performance. Furthermore, their costs should not exceed costs for network operation itself. Another problem is the overwhelming amount of result data we would get if we could afford to install specialized measurement hardware at each network node. Resource limitations are even worse in wireless networks where devices are smaller and therefore provide less resources. In addition to this, wireless transmission capacities are usually smaller than those for wired transmissions.

A second reason why we cannot measure everything have to do with *privacy concerns*. Users do not want to reveal too much information about their traffic. Providers are reluctant to share captured data with competitors. Plus the legal situation varies enormously from country to country making it even more more problematic if detailed data is shared among providers in different countries.

A third challenge is the analysis of *encrypted traffic*. We hope that the use of encryption techniques and secure protocols (e.g., IPsec) further increases in future. Encryption techniques limit attackers in gathering information or altering data exchanged on the network. Even so, the use of encryption also limits the ability of detection systems to analyze what is going on in the network and to model what can be considered as normal behavior. For some applications, it is possible to guess the application from packet patterns without deep packet inspection. However, this is quite challenging and does not work for arbitrary applications. To date, it is

unclear what bearing the inability to inspect encrypted traffic has on attack detection techniques.

We postulate the following requirements that a system should fulfill in order to establish situation awareness:

- Cope with resource constraints
- Change viewpoints
- Share information
- Respect privacy concerns

In the following we describe the requirements and the challenges derived from them more in depth.

4.2.1 Cope with Resource Constraints

The amount of data traffic carried by the Internet each day has increased dramatically over the last decades. A deceleration of this trend is not in sight. Technologies that allow higher data rates increase not just the amount of data that can be measured but also the quantity of measurement results needing to be processed, stored, or transferred per time unit.

Furthermore, the required measurement granularity can vary. Detection of some attacks requires fine-grained flow analysis, other attacks need deep packet inspection. So the main challenge is to keep up with high packet rates and increasing demands for fine grained information. Resources required for storage, post-processing, and transport of results increase with the amount of data captured per packet. In some cases additional information (e.g., arrival times, flow ID) also needs to be stored.

This can lead to an overwhelming amount of results data that can grow even larger than the actual data transmitted on the network itself, e.g., if the whole packet content is transmitted with additional information like timestamps. There are several different approaches to how measurement functions can cope with resource constraints.

One approach to dealing with increasing packet rates is the development and use of *specialized hardware* for measurement tasks. In [33] it is pointed out that design goals for standard PCs and network interface cards contradict measurement needs. One instance of this is that interface cards can arbitrarily discard packets if rates are too high, whereas for precise measurements it is important to capture all incoming packets. Furthermore, early and accurate time stamping of received packets is important for measurements but not available in standard PCs. The paper concludes that only specialized hardware can fulfill measurement demands.

In [17] Luca Deri describes the lessons learned from trying to use standard hardware for measurements in Gigabit networks. He recommends assessing measurement system performance in packets/second and not in Mbit, because it is more difficult to measure many small packets than a few large ones. He sees one main bottleneck in the file system performance that impedes storage of data at Gbit speed.

He also criticizes the “divide et impera” concept whereby traffic to be measured is distributed among several probes. This approach requires multiple probes and does not reduce the amount of results data. He rather suggests the use of traffic preprocessors such as the nProbe [17] which optimize packet capturing and flow cache operations. As a further possibility he mentions the use of sampling methods.

Endace Systems [25] provides hardware cards, the DAG boards [16], specialized in packet capturing and widely used within the research community. DAG boards are based on a configurable and programmable structure and include on-board filtering of data before it is passed to further processing. They allow highly accurate timestamping of packet arrivals with special support functions for the use of GPS-based clock synchronization. The fact that many scientists are using the same measurement hardware also increases consistency and comparability of measurement results.

The problem with solutions based on hardware is high costs. Especially if network wide measurements are required, dedicated hardware has to be placed at multiple observation points. This can increase the costs immensely.

Another approach for coping with increasing packet rates is the *improvement of algorithms* for packet processing (e.g., storage and classification). As packet processing is needed for a variety of network functions (e.g., routing, QoS provisioning), optimization of packet classification and filtering techniques has become a broad field of research. A well-known packet filter implemented in Unix systems is the Berkley Packet Filter (BPF). BPF is used by the broadly deployed tool tcpdump [47].

New packet filtering and classification techniques are described in [[59], [27], [64], [66], and [6]. These new techniques mainly focus on the improvement of search algorithms for finding the appropriate entry within classification rules and for the intelligent organization of data storage after classification. Feldmann and Muthukrishnan [31] describe the trade-offs between time and space requirements for packet classification. A method presented in [15] tries to reduce the required number of evaluated filter expressions operating at higher speeds. An overview of classification and filtering algorithms is given in [57].

Due to higher packet rates and fine grained flow definition, a higher number of flows can also be observed (e.g., [30]). More efficient methods are needed to store per flow information and to reduce the number of flow cache operations needed (e.g., lookup). Reference [37] describes a technique to reduce storage requirements by efficiently organizing per packet and per flow information to capture and store flow information on 10GB-Ethernet and OC-192 links. Space-Code Bloom Filter (SCBF) is another approach for efficient storage [41].

The optimization of algorithms is useful for coping with higher data rates, but it has limitations. This research field has been active for quite some time yet it is difficult to find further significant improvements. What’s more, if algorithms are implemented in hardware they usually lack the flexibility and re-configurability that software-based solutions can offer.

The third approach for coping with ever higher data rates is to apply *data selection techniques*. As early as 1989 Paul Amer and Lillian Cassel proposed the use

of packet sampling techniques for real-time status reporting in IP networks [2]. Their paper describes sampling-based measurements of peak load, packet rates and changes in those metrics. Two different sampling techniques are introduced, a systematic and a random method with a time-based start trigger and a count-based stop trigger, realized by the enabling and disabling of the receive function of the measurement device.

A much cited early work on sampling was published in 1993 by Claffy [12]. She and her co-authors describe the empirical investigation of different sampling schemes for the estimation of distributions of packet sizes and interarrival times. In their work five different sampling schemes are compared: Count- and time-based systematic sampling, count- and time-based stratified sampling and a count-based n -out-of- N sampling. In [22] Duffield addresses the influences on flow reporting, resource consumption, and the accuracy for volume estimations based on sampling techniques.

In recent years, researchers have also used sampling for the estimation of more unusual metrics such as the temporal characteristics or spectral density of the packet arrival process or the tracking of the path a packet takes through the Internet. The authors of [50] focus on the detection of temporal correlations in a trace. They introduce a sampling method called Fast Correlation-Aware Sampling (FastCARS). This method consists of a superposition of multiple systematic count-based sampling processes (in their paper this is called deterministic event-driven sampling). The authors in [36] aim at the recovery of the spectral density of the packet arrival process and the distribution of the number of packets per flow from sampled values.

An overview of selection techniques is given in [20]. Basic packet selection methods such as probabilistic sampling, systematic time- and count-based sampling, and random n -out-of- N sampling are currently standardized in the IETF PSAMP group [70]. Besides various packet selection techniques, flow selection techniques (e.g., [21]) and combined methods (e.g., sample-and-hold [29]) have also been proposed in the literature.

One problem with data selection techniques is that they substitute the exact measurement of the metric of interest by an estimate of the said metric. An estimate is useless if we cannot assess estimation accuracy. This means that an accuracy statement has to be provided with the measurement result in order to assess the estimation error. This is no easy task as the accuracy often depends on the parent population. In our case this is the traffic in the network, which is highly dynamic so that the accuracy statement has to be continuously updated. Furthermore, the accuracy statement has to be calculated based on the selected data only. Due to storage limitations it is sometimes even based on an aggregated version of this data [69].

Besides the high resource consumption in general, control of the resource consumption poses yet another problem. Passive measurements rely on the traffic existent in the network so that the amount of resources required for measurements also vary. As such traffic is highly dynamic, we can observe changes in packet rates and the number of flows, giving us extremely variable resource demands. To cope with this some approaches have been proposed for controlling resource consumption by using adaptive data selection or aggregation techniques. Another contrary goal is to

aim at a constant estimation accuracy and to adapt selection techniques to this end. Adaptive data selection techniques are described in the section below.

4.2.2 Change Viewpoints

The ability to change viewpoints is extremely valuable for establishing situation awareness. In order to generate a good picture of the current situation, it is useful to have the option of zooming in or out. Sometimes it is sufficient to look at link counters, or coarse grained flow information. At other times, e.g., if one suspects an attack, enabling deep packet inspection might be required to further analyze what is happening. Some commercial intrusion detection systems use a multistage approach where suspicious traffic is re-directed to a device that performs an in-depth analysis.

The capability to re-configure the measurement system provides the basis for *adaptive measurement* techniques and is a prerequisite for resource and accuracy control. Adaptive measurements can be used to tune toward events of interest by changing observation points, classification rules, or aggregation and selection techniques on demand. IBM has recently proposed “zoom monitors” that adapt information granularity based on the relevance of the traffic by re-configuring the measurement system [62].

Initial approaches for adaptive data selection aimed at maintaining a constant resource consumption. In [19] an adaptive sampling scheme is presented that aims at a constant resource consumption to improve utilization of available CPU power.

Further approaches aim at a stable estimation accuracy. Choi introduces an interesting concept of adaptive sampling for the detection of changes in traffic load [11]. She uses time-based measurement intervals and a random probabilistic sampling for packet selection. The goal here is to keep the estimation accuracy (error and confidence level) within given boundaries. This is achieved by adapting the sampling probability within an observation period to the expected traffic characteristics of that interval.

The authors of [28] propose extending Cisco NetFlow to allow an adaptation of the sampling rate. This is motivated by the fact that resource consumption, memory, and bandwidth required to store and transport flow records, depends largely on the number and type of arriving packets at the observation point. They especially point out that flow cache requirements and the number of flow records can mushroom when a flooding attack hits the router, introducing additional load problems for the router under attack. This can be prevented by using the proposed adaptive NetFlow which reduces the sampling rate if the packet rate increases.

4.2.3 Share Information

Information sharing is the key for successful defense. The advance to be made by combining information from multiple (trusted) parties is immense. Sharing information is the prerequisite for learning from others. If one node observes suspicious traffic it is useful to see whether other nodes have observed similar phenomena. If a node is infected by a virus or a worm that spreads within a network, it is worthwhile

checking whether neighbor nodes or neighbor networks have experienced similar events in the past. If this is the case, information from neighbors can help to analyze the attack, select appropriate countermeasures, or nip it in the bud.

There are different levels of information sharing. The first level is the sharing of raw measurement data. One method for sharing observations from different network nodes is the realization of *multi-point measurements*, i.e., the collection of data from multiple observation points. Multi-point measurements are essential for gaining a network-wide view. They are also a prerequisite for calculating certain metrics. By observing the same packet from multiple observation points we can calculate packet delays or the path the packet took through the network. Sharing post-processed, derived data or local decisions from different network nodes is also useful. Different nodes may come to a different conclusion about the current threat level in the network. Generating a joint decision from this requires some coordination, but can improve detection quality.

Today there are various tools available that support multi-point measurement. But correlation of data from multiple observation points still holds some challenges. Correlating time information from different observation points requires clock synchronization. Different approaches exist, such as the network time protocol (NTP) or GPS-based systems. But nowadays most routers still do not support accurate clock synchronization systems that would for instance allow network-wide passive one-way-delay measurements.

Furthermore, packet events that occur at different observation points need to be correlated. This can be done by using parts of the packet content to recognize the packet at different points in the network. But this requires storage and transmission of the packet content of all measured packets. To save resources, it has been proposed to transfer a packet ID instead of the whole packet to identify packet arrivals at different points in the network (e.g., [35], [23], [71]).

One quite difficult case is the combination of data selection techniques with multi-point measurements. It needs to be ensured that the same packet is selected at different points. But packet sequence and arrival times differ at the observation points so that the selection of the same packet can be only guaranteed with filtering techniques which operate on parts of the packet content. Sampling techniques that operate on arrival time or packet sequence are not applicable. In [23] and [70] hash-based selection techniques are discussed that aim at emulation of random sampling.

Analysis of data from multiple observation points can be done in a centralized or decentralized way. In a decentralized case this would require each network node to request information from other network nodes in order to establish their own situational view. With this, each node can request the required input for decision-making processes on the node.

In some cases it is also useful to incorporate information from different network layers (cross-layer approaches). Further information about the network environment (e.g., geo-location of network nodes), about user behavior or external events (e.g., content has been placed on the web that will attract many users) can be useful

to build a situational view while incorporating information from other instances in the network (e.g., information from a AAA server) can help to further analyze the situation and improve defense strategies [68].

A more difficult challenge than sharing information from different observation points in one network is sharing information among network operators. Data exchange between providers can help to better identify an attack, track an attacker faster, and isolate the source of the attack. But operators are extremely reluctant to share data about the traffic in their network because such data may reveal information about the network structure, the users or network vulnerabilities. One main fear is that competitors might get information they can use for their own advantage. Another should be that potential attackers could learn about the network and its traffic and so are able to craft better attacks. Privacy concerns also limit the ability to share information.

Besides the disclosure of information problem, there is also an architectural challenge when sharing information. There are other areas where information sharing among providers is necessary for network operation. One example is the border gateway protocol (BGP) used to share routing information among border routers. BGP demonstrates that providers have found solutions to share data. But it also shows the immense problems that can arise from this. To give but one example, configuration errors introduced by unexperienced providers can have an impact on a global scale.

In addition, communication to improve attack detection by cooperation can induce major vulnerability of the whole detection system. If an attacker manages to break into the communication path he or she can simply pretend that an attack is going on by simply reporting incidents. Thus the resources of the whole detection system may be bound up, without even the need for a real attack. Communication methods that help in protecting the network have themselves to be well protected against attacks.

Another aspect is that the research community is starved for data. Scientists who have data (e.g., from projects with operators) are not allowed to publish or share it. This is especially true if data contains attacks. Consequently, there is no useful reference data for investigating and comparing attack detection algorithms. This is an immense constraint and impediment to research on detection methods. Hopefully it will not require a super attack to recognize the advantage of sharing information and to speed up research for joint network protection.

Sharing information requires standardized interfaces. In January 2008 the IETF standardized the IP Flow Information Export (IPFIX) protocol [13] for exporting flow information from routers and network probes. This protocol can be also used for exporting packet information or derived data. For storing and sharing packet information the tcpdump packet file format is often used as a de facto standard. For storing flow information the IPFIX group discusses a proposal for an IPFIX file format [63]. Standardized formats for the storage of measurement data also help researchers to share and compare data and give an incentive to provide tools with standardized interfaces.

4.2.4 Respect Privacy

The need to respect privacy concerns is often in contradiction with the desire to get as much information as possible. Privacy concerns need to be respected but they do constrain data collection and information sharing.

The legal situation and public perceptions on the need to protect data both vary from country to country. There is a need to investigate methods that are able to detect attacks without compromising user privacy. Methods that do not require deep packet inspection are of advantage. Furthermore, techniques such as aggregation or sampling can conceal user information details so methods that can work with highly aggregated or sampled data may be less problematic. However, given the increase of higher-layer and ever more sophisticated attacks, it is often impossible to rely for detection solely on coarse grained or aggregated data.

The situation is less critical as long as the data does not leave the provider network. It becomes much more severe when raw data is shared among providers or researchers. Use of anonymization techniques can be of assistance. Even so, anonymization has to keep the analysis goal in mind. This means it has to be ensured that the aspects of interest can still be analyzed so not every technique is applicable to each case.

An alternative discussed in the research community, is to bring the analysis software to the data instead of sending data to interested researchers. This involves researchers sending their analysis scripts to the data owner. The data owner then runs the scripts on the data and only sends back the analysis results.

4.3 Decision Making

Decision making is the next step in the decision cycle. For self-protection, decision making mainly concerns decisions about threat level and counter actions. It needs to be decided whether the system suspects an attack or a failure that requires immediate action.

One challenge for the decision-making process is to select the right information of relevance to the decision. A further challenge from the measurement side is to provide the relevant information in time. If the information arrives too late it might be useless for the decision-making process.

The decision-making process should be able to control situation awareness. This means it should be able to re-configure measurement and analysis functions in order to change classification rules, increase or decrease sampling and control the calculation of statistics and derived metrics. The ideal case when all required information is available in time is quite rare. Decision-making processes therefore need to deal with situations where they have incomplete data and need to decide in uncertainty.

The decision-making process should incorporate learning strategies to improve the decision quality. Furthermore, prediction techniques can be used to understand and detect a potentially negative evolving situation and to analyze alternative decisions and their potential impact.

Decision making for self-protection requires attack detection methods. We distinguish two attack detection methods based on traffic observation. *Signature-based detection* is the recognition of previously captured attack patterns. Such methods require knowledge about the expected attack traffic and therefore only work for known attacks. A second method is *anomaly detection*. Anomaly detection tries to detect deviations from the expected normal behavior. Anomaly detection can also detect unknown attacks (zero-day events). However, it relies on a good knowledge and prediction of what is considered as normal behavior. The decision whether traffic patterns indicate an attack or not can turn into a tough challenge as attackers increasingly adapt their strategies to detection systems and try to conceal attack traffic by using so-called stealth attacks.

4.3.1 Signature-Based Detection

Signature-based detection techniques are based on the recognition of a known signature. Observed traffic characteristics are compared to a previously stored pattern. The pattern represents a traffic characteristic that is typically produced by attack or attack preparation activities. If the observed traffic and the stored pattern are equal, the system sends an alarm. Since it is known what the system is looking for, parameters for traffic observation and analysis can be set statically. Signature-based detection techniques are used in virus scanners, or intrusion detection systems such as SNORT [58].

As attacks become more sophisticated, signatures to detect attacks become more complex. It often requires deep packet inspection or the re-assembly of bidirectional communication patterns. Thus one challenge for signature-based detection is the fast analysis of traffic patterns and the high amount of data that needs to be analyzed. Another problem is that attack patterns change and signatures must be updated as new attack patterns become known. The biggest drawback of signature-based detection techniques is that they only work on known or predictable attacks.

4.3.2 Anomaly Detection

Anomaly detection is based on the detections of deviations from the normal situation. Usually a model is built representing the normal state. Traffic observations are then compared to the normal network operation. If deviations to normal behavior exceed a given threshold, the system sends an alarm. Thus anomaly detection systems can also detect new attacks. Anomaly detection can be based on statistical data analysis, machine learning, or data mining techniques. An overview of anomaly detection systems and the various approaches is given in [51].

4.3.3 Collaborative Attack Detection

Collaborative Attack Detection is based on information sharing among detection algorithms. In the same way as information sharing at the measurement layer increases situation awareness, one can improve the situational view by sharing

analysis results among detection systems. Here we should distinguish two different aspects.

In the *Intra-domain* case detection methods or partial decision-making processes run on different nodes in one network. Nodes then share their local view of the situation and their analysis results. Based on the shared information a joint detection decision has to be formed. Distributing the detection task among multiple nodes also provides a method of sharing the processing load. Since the same packets can be observed at multiple nodes in one network, each node can look at different aspects, e.g., observe different flows or selected traffic at a finer granularity.

Some detection systems already distribute the detection task. In such systems the whole traffic is analyzed in a coarse grained analysis. Only suspicious traffic is re-directed to a more fine grained analysis. Nowadays, detection tasks are often performed by dedicated hardware. Sharing detection resources among standard routers is a potential alternative if the subtasks become less resource intensive.

The decision making can be done in a centralized or decentralized way. In a *centralized* approach the analysis results and local decisions are collected and evaluated on a central server. This has the advantage that only one instance needs to collect the information from all participating nodes. Furthermore, only the central instance makes the final decision and invokes the counteractions if required so there is no need for further coordination this with all the nodes. All this decreases the complexity of the system while also reducing the communication demands and making the detection system less vulnerable to attacks on its internal communication.

On the other hand, a central server entails the risk that a failure or a targeted attack on the server endangers the operation of the whole system. In a *decentralized* detection scenario, network nodes themselves should build to a joint decision. This can be done in a hierarchical manner with dedicated roles per node, by selecting a leader on demand or by using voting or other methods to make a joint decision among equal members of a group.

Organizing such a distributed decision process is no easy matter. First of all we have to deal with the inter-operation of multiple control loops. Policies for the cooperation of these loops have to be carefully selected to prevent undesired states, deadlocks, or oscillations and communication demands are also higher. This means not only more traffic in the network but also more vulnerabilities to failures or attacks against the internal node communication. Establishing trust among the nodes or encrypting the exchanged information are approaches for improving resistance against attacks on node communication.

Another aspect is timely access to the required information. The time that is required for traffic observation and analysis, decision making, and reconfiguration has to be considered when the control loops are designed. Decision-making algorithms need fallback solutions for cases where the required information is not available at the time where the decision is needed. Participation of multiple nodes in the decision-making process increases interdependencies and makes the overall process much more complex.

The challenges for distributed collaborative decision making can be summarized as follows:

- Setting collaboration policies
- Delivering the required information in time
- Protecting internal communication

A second level for a collaborative attack detection is the *Inter-domain* case, in which detection systems of different networks share their views about the situation and cooperate in detection. Networks can share signatures from attacks they have experienced in the past. If available, successful defense strategies can also be exchanged. Warning neighbors about suspicious activities in the own network can provide them with a valuable time advantage to prepare defense strategies before the attack spreads into their network. In addition, anomaly detection systems can profit enormously from getting data from neighbors as input to learning processes for modeling normal network behavior.

Even so, as mentioned above information sharing among providers is very difficult. This is not only the case for sharing measurement information. Even information exchange about incidents is not very common. Quite often incidents are not reported, because providers fear negative publicity. This is one of the negative effects of competition as it prevents learning. As incidents with destruction on a global scale regularly show, attackers are ahead of the game.

The IETF working group on Extended Incident Handling (INCH) made an attempt to standardize a format to exchange information about computer security incidents. This was intended to improve the work of the Computer Security Incident Response Teams (CSIRTs). The group developed an Incident Object Description Exchange Format (IODEF). An attempt was also made to standardize a Real-time Inter-Network Defense (RID) protocol for enabling inter-network defense strategies by joint incident handling. However, due to a lack of supporters, the INCH working group was closed in October 2006. Some implementations for sharing information do exist (e.g., Automated Incident Reporting AirCERT [1]) and Sprint has made an attempt to standardize an architecture for data sharing among operators, but they failed when proposing to build an IETF working group around this topic. It seems that the majority of operators is not yet sensitized enough to recognize the importance of sharing information.

4.4 Re-Configuration

Self-protection requires the ability to change network configuration in order to react to attacks. This includes the ability to enable filtering techniques to block unwanted traffic. The ability to influence routing is also useful to re-direct traffic or eliminate infected nodes. As explained above, it is also needed to allow measurement re-configuration, e.g., to support a more in depth analysis of suspicious traffic.

The most common techniques for reacting to an attack are the following:

- Traffic blocking
- Traffic redirection
- Elimination of infected systems or services

Nowadays network devices are often configured manually using proprietary vendor-specific languages and protocols. In December 2006 the IETF working group on network configuration (NETCONF) standardized a protocol for the configuration of network devices [26]. The protocol can use Secure Shell (SSH), Blocks Extensible Exchange Protocol (BEEP), or the Simple Object Access Protocol (SOAP) as the underlying transport. The different uses of the various transport protocols are described in [65], [34], and [42]. An approach to secure NETCONF data exchange by Transport Layer Security (TLS) is described in [3].

The IPFIX group has already adopted the concept to allow re-configuration of measurement devices and data export. A data model for configuring IPFIX and PSAMP processes is proposed in [49]. It uses the data modeling language YANG. YANG has been proposed as the data modeling language for NETCONF in [5].

4.5 Protection of the Detection System

Protecting the detection system itself is extremely important. If adversaries can trick the detection system it is not even necessary to launch a real attack to disturb network operation. Loading the system with wrong alarms or triggering the activation of unnecessary counteractions can decisively influence network operation.

A protection system based on information sharing and learning introduces new vulnerabilities. New vulnerabilities can arise from the following features:

- *Communication among nodes*: Sharing information and trust establishment requires additional communication paths. If an attacker manages to interrupt communication or to alter the sent information it is possible to trick the detection system.
- *Adaptation techniques*: Control loops used for automatic adaptation can be destroyed or overloaded by attackers.
- *Learning techniques*: If learning techniques are used and information about the learning process can be discovered, an attacker can train the system to consider attack behavior as normal.
- *New software*: The introduction of new methods for self-protection always entails the risk of new vulnerabilities. Extensive testing is required before new methods can be relied on.

Much more research is required in this area and in the area of autonomic communication in general to prevent attacks on newly deployed methods in future networks.

5 Conclusion

Self-protection concepts incorporate decision-making cycles for network security directly into the network. Based on the principles of autonomic communication, they aim to reduce human intervention for attack prevention, detection, and defense. Prevention mechanisms at network level can be supported by the incorporation of prevention techniques in service composition. Detection and defense should incorporate further functions in the network for making decisions on threat level and counteractions.

All this is predicated on the establishment of situation awareness which can be achieved by the monitoring and analysis of traffic and environment. Given the dynamic nature of network traffic this is a challenging and costly task. Cooperation of network nodes during observation and analysis is useful for improving situation awareness.

The decision-making process within the network can be even more challenging. Limited by the achievable situation awareness and timing requirements, decisions have to be made in uncertainty based on incomplete and untimely delivered information. Decisions have to be made fast and with consideration of the costs and benefits arising from the decision. Learning techniques provide a basis to improve decision-making. Cooperation of nodes during decision making helps to improve decision quality, but also introduces further challenges such as establishing suitable communication among nodes and the handling of multiple concatenated control loops.

In order to (re-)configure network nodes with regard to the decision outcome, it is necessary to provide flexible network functions and standardized configuration interfaces.

Many building blocks for self-protection already exist, such as measurement software, protocols for node cooperation, adaptation techniques, learning algorithms, and configuration interfaces. Nevertheless, the integration and adjustment of such techniques to enable collaborative detection and joint decision making in future networks still remains a daunting challenge.

References

1. AirCERT. <http://aircert.sourceforge.net/>.
2. P. D. Amer and L. N. Cassel. Management of sampled real-time network measurements. In *14th Conference on Local Computer Networks*, October 1989.
3. M. Badra. NETCONF over Transport Layer Security (TLS), February 2008. Internet Draft, work in progress.
4. S. Bansal and M. Baker. *Observation-Based Cooperation Enforcement in ad hoc Networks*, 2003.
5. M. Bjorklund (ed.). YANG – A data modeling language for NETCONF, January 2009. Internet Draft, work in progress.
6. H. Bos, W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis. FFPF: Fairly Fast Packet Filters. In *Proceedings of OSDI'04*, 2004.

7. S. Buchegger and J.-Y. L. Boudec. Performance analysis of the CONFIDANT protocol: Cooperation of nodes—fairness in dynamic ad-hoc networks. In *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Lausanne, Switzerland, June 2002. IEEE.
8. J. Buford, R. Kumar, and G. Perkins. Composition trust bindings in pervasive computing service composition. In *PERCOMW '06: Proceedings of the 4th annual IEEE international conference on Pervasive Computing and Communications Workshops*, p. 261, Washington, DC, USA, 2006. IEEE Computer Society.
9. L. Buttyán and J. Hubaux. *Nuglets: A Virtual Currency to Stimulate Cooperation in Self-Organized ad hoc Networks*. Technical Report DSC/2001, 2001.
10. D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha. Service composition for mobile environments. *J. Mobile Netw. Appl., Special Issue on Mobile Services*, 10(4): 435–451, January 2005.
11. B.-Y. Choi, J. Park, and Z.-L. Zhang. Adaptive random sampling for load change detection. *SIGMETRICS Perform. Eval. Rev.*, 30(1): 272–273, 2002.
12. K. C. Claffy, G. C. Polyzos, and H. W. Braun. Application of sampling methodologies to network traffic characterization. In *ACM SIGCOMM*, pp. 194–203, 1993.
13. B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008.
14. E. M. J. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts and London, England, 1999.
15. J. Coppens, S. D. Smet, S. V. den Berghe, F. D. Turck, and P. Demeester. Performance evaluation of a probabilistic packet filter optimization algorithm for high-speed network monitoring. In *HSNMC*, pp. 120–131, 2004.
16. The DAG project. <http://dag.cs.waikato.ac.nz>.
17. L. Deri. nprobe: an open source netflow probe for gigabit networks. In *Proc. of Terena TNC2003*, 2003.
18. S. Dobson, S. Denazis, A. Fernández, D. Gäiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2): 223–259, 2006.
19. J. Drobisz and K. J. Christensen. Adaptive sampling methods to determine network traffic statistics including the hurst parameter. In *LCN*, pp. 238–248, 1998.
20. N. Duffield. Sampling for passive internet measurement: A review. In *Statistical Science*, Vol. 19, pp. 472–498, 2004.
21. N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *Proc. Internet Measurement Workshop*, November 2001.
22. N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *ACM SIGCOMM Internet Measurement Workshop*, 2002.
23. N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *SIGCOMM*, pp. 271–282, 2000.
24. S. Dustdar and W. Schreiner. A survey on web services composition. *IJWGS*, 1(1): 1–30, 2005.
25. Endace measurement systems. <http://www.endace.com>.
26. R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006.
27. F. Ergun, S. Mitra, S. C. Sahinalp, J. Sharp, and R. K. Sinha. A dynamic lookup scheme for bursty access patterns. In *INFOCOM*, pp. 1444–1453, 2001.
28. C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better NetFlow. In *SIGCOMM*, 2004.
29. C. Estan and G. Varghese. New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3): 270–313, 2003.
30. W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Global Telecommunications Conference*, December 1999.

31. A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *INFOCOM*, pp. 1193–1202, 2000.
32. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
33. C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. *Lecture Notes in Computer Science*, 2170: 556+, 2001.
34. T. Goddard. Using NETCONF over the Simple Object Access Protocol (SOAP). RFC 4743 (Proposed Standard), December 2006.
35. I. D. Graham, S. F. Donnelly, S. Martin, J. Martens, and J. G. Cleary. Nonintrusive and accurate measurement of unidirectional delay and delay variation on the internet. In *INET*, 1998.
36. N. Hohn and D. Veitch. Inverting sampled traffic. ACM SIGCOMM internet measurement conference (IMC 2003), Miami Beach, Florida, USA, October 2003.
37. G. Iannaccone, C. Diot, I. Graham, and N. McKeown. Monitoring very high speed links. In *ACM Internet Measurement Workshop*, 2001.
38. IBM. An architectural blueprint for autonomic computing. white paper, IBM, 2006.
39. C. Jacob, H. Pfeffer, L. Zhang, and S. Steglich. Establishing service communities in peer-to-peer networks. In *1st IEEE International Peer-to-Peer for Handheld Devices Workshop CCNC 2008*, Las Vegas, NV, USA, January 10–12 2008.
40. J. Koehler, C. Giblin, D. Gantenbein, and R. Hauser. *On Autonomic Computing Architectures*, 2003.
41. A. Kumar, J. Xu, J. Wang, O. Spatscheck, and L. Li. Space-code bloom filter for efficient per-flow traffic measurement. In *Infocom*, 2004.
42. E. Lear and K. Crozier. Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP). RFC 4744 (Proposed Standard), December 2006.
43. D. Linner, H. Pfeffer, A. Kress, S. Kruessel, and S. Steglich. SmartWare, 2008.
44. D. Linner, H. Pfeffer, I. Radosch, and S. Steglich. Biology as Inspiration towards a new Service Life-Cycle. In *Proceedings of the 4th IEEE International Conference on Autonomic and Trusted Computing (ATC'07)*, ISBN: 978-3-540-73546-5, pp. 94–102, Hong Kong, China, July 11–13 2007.
45. D. Linner, H. Pfeffer, and S. Steglich. A genetic algorithm for the adaptation of service compositions. In *Proceedings of the 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2007.
46. D. Linner, I. Radosch, S. Steglich, and C. Jacob. The semantic data space for loosely coupled service provisioning. In *ISADS '07: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems*, pp. 97–104, Washington, DC, USA, 2007. IEEE Computer Society.
47. S. McCanne, V. Jacobson, C. Leres. tcpdump manual page, 2001. Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, USA.
48. B. Miller. *The Autonomic Computing Edge: Can you Chop Up Autonomic Computing?* Technical report, IBM, 2008. available at <http://www.ibm.com/developerworks/autonomic/library/ac-edge4/>.
49. G. Muenz and B. Claise. Configuration Data Model for IPFIX and PSAMP, November 2008. Internet Draft, work in progress.
50. J.-Y. Pan, S. Seshan, and C. Faloutsos. FastCARS: Fast, Correlation-Aware Sampling for Network Data Mining. In *Proceedings of IEEE GlobeCOM 2002 – Global Internet Symposium*, 2002.
51. A. Patcha and J.-M. Park. An overview of anomaly detection techniques: existing solutions and latest technological trends. *Comput. Netw.*, 51(12): 3448–3470, 2007.
52. H. Pfeffer, D. Linner, I. Radosch, and S. Steglich. The bio-inspired Service Life-Cycle: an overview. In *Proceedings of the 3rd IEEE International Conference on Autonomic and Autonomous Systems (ICAS'07)*, Athens, Greece, June 19–15 2007.

53. A. Pietzowski, B. Satzger, W. Trumler, and T. Ungerer. A bio-inspired approach for self-protecting an organic middleware with artificial antibodies. In *IWSOS/EuroNGI*, pp. 202–215, 2006.
54. A. Pietzowski, B. Satzger, W. Trumler, and T. Ungerer. Using positive and negative selection from immunology for detection of anomalies in a self-protecting middleware. In *36th annual conference of the Gesellschaft für Informatik e.V. (GI), Informatik für Menschen, INFORMATIK 2006*, Vol. P-93 of LNI, Dresden, Germany, October 2006.
55. A. Pietzowski, W. Trumler, and T. Ungerer. An artificial immune system and its integration into an organic middleware for self-protection. In M. Cattolico (ed.), *GECCO*, pp. 129–130. ACM, 2006.
56. J. Rao and X. Su. A survey of automated web service composition methods. In *SWSWPC*, pp. 43–54, 2004.
57. C. Schmoll. Dynamically configurable network meter for accounting in ip-based networks. Diploma thesis, Technical University Berlin, December 2001.
58. SNORT. <http://www.snort.org/>.
59. V. Srinivasan. A packet classification and filter management system. In *INFOCOM*, pp. 1464–1473, 2001.
60. R. Sterritt and D. Bustard. Towards an autonomic computing environment. In *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, p. 699, Washington, DC, USA, 2003. IEEE Computer Society.
61. R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A concise introduction to autonomic computing. *Advanced Engineering Informatics*, 19(3): 181–187, 2005.
62. M. P. Stoecklin, A. Kind, and J.-Y. L. Boudec. Dynamic adaptation of flow information granularity for incident analysis. In *CERT FloCon Workshop*, 2008.
63. B. Trammell, E. Boschi, L. Mark, T. Zseby, and A. Wagner. An ipfix-based file format, October 2008. Internet Draft, work in progress.
64. P. R. Warkhede, S. Suri, and G. Varghese. Fast packet classification for two-dimensional conflict-free filters. In *INFOCOM*, pp. 1434–1443, 2001.
65. M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742 (Proposed Standard), December 2006.
66. T. Y. C. Woo. A modular approach to packet classification: Algorithms and results. In *INFOCOM*, pp. 1213–1222, 2000.
67. S. Zhong, Y. Yang, and J. Chen. *Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile ad hoc Networks*, 2002.
68. T. Zseby, E. Boschi, N. Brownlee, and B. Claise. IPFIX Applicability, June 2007. Internet Draft, work in progress.
69. T. Zseby, T. Hirsch, and B. Claise. Packet sampling for flow accounting: challenges and limitations. In *Ninth Passive and Active Measurement conference (PAM)*, April 2008.
70. T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall. Sampling and Filtering Techniques for IP Packet Selection, July 2008. Internet Draft, work in progress.
71. T. Zseby, S. Zander, and G. Carle. Evaluation of building blocks for passive one-way-delay measurements. In *Proceedings of Passive and Active Measurement Workshop (PAM 2001)*, April 2001.

Formal Aspects of Self-* in Autonomic Networked Computing Systems

Phan Cong-Vinh

Abstract A new computing paradigm is currently on spot: autonomic computing (AC), which is inspired by the human autonomic nervous system. AC is characterized by its self-* facets such as self-configuration, self-healing, self-optimization, and self-protection. The overarching goal of AC is to realize computer systems, and thus networked computing systems, that can manage themselves without direct human interventions. Meeting this grand challenge of autonomic computing requires a fundamental approach to the notion of self-*. To this end, taking advantage of the categorical approach we establish, in this chapter, a firm formal basis for modeling self-* in autonomic networked computing systems, developing self-* monoid, category of self-* monoids, and series of self-* facets. All of these are to achieve formal aspects of the self-*.

1 Introduction

Networked computing is a characteristic of many modern computing systems and implies an increased complexity in managing the system behavior. *Autonomic computing* (AC) is essential to keep such systems manageable. In fact, the problem is that many networked computing systems make central or global control impossible. For example, the information needed to make decisions cannot be gathered centrally (e.g., the type of mobile ad hoc networks (MANETs)). In such the networked computing systems, AC is only possible when networked computational entities autonomously interact and coordinate with each other to maintain properly the required computations. Therefore, in the networking environments, we denote AC as *autonomic networked computing* (ANC). In other words, when AC mechanism is

P. Cong-Vinh (✉)

Centre for Applied Formal Methods, London South Bank University, Borough Road, London SE1 0AA, United Kingdom
e-mail: phanvc@ieee.org

implemented in the networked computing systems then it defines ANC paradigm. The essence of ANC is to enable the autonomic networked computational entities to govern themselves the set of services and resources delivered at any given time while interacting and coordinating with each other. Hence, for *ANC systems* (ANCSs), one of major challenges is how to support self-governance in the face of changing user needs, environmental conditions, and computation objectives. In other words, how does an ANCS understand relevant contextual data, change to those data and adapt the services and resources, which it provides, in accordance with goal-driven computational mechanisms?

Dealing with this grand challenge of ANCSs requires a well-founded modeling and in-depth analysis on the notion of ANC. With this aim, we develop a firm formal approach in which autonomic networked computational entities are able to detect, diagnose and repair faults, as well as adapt their configuration and optimize their performance in the face of changing user needs and environmental conditions. All of these must be done while protecting and healing themselves in the face of natural problems and malicious attacks.

AC is often described as self-*, but ANC focuses on self-knowledge in preference to build self-governance. However, self-* functionality is still supported, but the emphasis of ANC is on the foundation to realize self-*, not in the different self-* technologies.

In this view, we see that rigorously approaching to ANC requests fundamental research in all aspects of the self-*. As a novel development for the self-*, we consider to formalize aspects of the self-* taking advantage of categorical language, whose content is presented in this chapter.

2 Outline

The chapter is a reference material for readers who already have a basic understanding of ANCS and are now ready to know the novel approach for formalizing self-* in ANCS using categorical language.

Formalization is presented in a straightforward fashion by discussing in detail the necessary components and briefly touching on the more advanced components. Several exercises and notes explaining how to use the formal aspects, including justifications needed in order to achieve the particular results, are presented.

We attempt to make the presentation as self-contained as possible, although familiarity with the notion of self-* in ANCS is assumed. Acquaintance with the algebra and the associated notion of categorical language is useful for recognizing the results, but is almost everywhere not strictly necessary.

The rest of this chapter is organized as follows: Sections 3, 4 and 5 present the notions of AC, ANC, and some categorical terms, respectively. Section 6 presents models of self-* in ANCSs. In Section 7, structures of self-* including self-* monoid, a category of self-* monoids and some algebraic properties are developed. Section 8 is a place to develop series of self-* facets in detail. In Section 9, we briefly discuss an alternative approach and compare it with our development. Finally, a short summary is given in Section 10.

3 Autonomic Computing as Self-*

AC imitates and simulates the natural intelligence possessed by the human autonomic nervous system using generic computers. This indicates that the nature of software in AC is the simulation and embodiment of human behaviors, and the extension of human capability, reachability, persistency, memory, and information processing speed [52]. AC was first proposed by IBM in 2001 where it is defined as

“Autonomic computing is an approach to self-managed computing systems with a minimum of human interference. The term derives from the body’s autonomic nervous system, which controls key functions without conscious awareness or involvement” [22].

AC is generally described as self-*. Formally, let self-* be the set of self-_'s. Each self-_ to be an element in self-* is called a *self-* facet*. That is,

$$\text{self-*} = \{\text{self-}_- \mid \text{self-}_- \text{ is a self-* facet}\} \tag{1}$$

We see that self-CHOP is composed of four self-* facets of self-configuration, self-healing, self-optimization, and self-protection. Hence, self-CHOP is a subset of self-*. That is, self-CHOP = {self-configuration, self-healing, self-optimization, self-protection} \subset self-*. Every self-* facet must satisfy some certain criteria, so-called *self-* properties*. In [55], Wolf and Holvoet classified the self-* properties in autonomic networks.

In its AC manifesto, IBM proposed eight facets setting forth an AC system (ACS) known as *self-awareness, self-configuration, self-optimization, self-maintenance, self-protection (security and integrity), self-adaptation, self-resource-allocation, and open-standard-based* [22]. Kinsner pointed out that these facets indicate that IBM perceives AC is a mimicry of human nervous systems [27]. In other words, self-awareness (consciousness) and non-imperative (goal-driven) behaviors are the main features of ACSs [52].

4 Autonomic Networked Computing

From the notion of AC, an ACS is defined by Wang as

“An autonomic computing system is an intelligent system that implements nondeterministic, context-dependent, and adaptive behaviors based on goal- and inference-driven mechanisms” [53].

This definition is concerned with three major factors of ACS:

- *Variable events*: AC systems do not rely on instructive and procedural information, but are dependent on variable events of ever-changing external environment and internal status formed by the long-term historical events.
- *Variable behaviors*: AC systems behave in a nondeterministic, context-dependent, and adaptive manner.

- *Goal-driven mechanisms*: AC systems do not rely on imperative and procedural instructions, but are dependent on goal-, perception-, and inference-driven mechanisms.

Consequently, ANC, and thus self-*, is achieved when an ACS is constructed as a group of locally interacting autonomous computational entities that cooperate in order to adaptively maintain the desired system-wide behavior without any external or central control. Such an ACS is viewed as an ANCS.

The topic of AC has seen a number of developments through various research investigations following the IBM initiative such as AC paradigm in [9, 18, 37, 41, 47]; different approaches and infrastructures in [1, 5, 44, 46, 55] for enabling autonomic behaviors [48–51]; core enabling systems, technologies, and services in [15, 16, 45, 3, 21, 31] to support the realization of self-* properties in autonomic systems and applications; specific realizations of self-* properties in autonomic systems and applications in [8, 13, 24, 20, 26, 34, 39]; architectures and modeling strategies of autonomic networks in [17, 33, 32]; middleware and service infrastructure as facilitators of autonomic communications in [11, 35, 19]; approaches in [12, 4, 40] to equipping current networks with autonomic functionality for migrating this type of networks to autonomic networks.

Moreover, AC has also been intensely studied by various areas of engineering including artificial intelligence, control systems, and human-orientated systems [25, 36, 53, 54]. Autonomic computing has been set as an important requirement for systems devised to work in new generation global networked and distributed environments such as wireless networks, P2P networks, Web systems, multi-agent systems, and grids [10, 12, 28, 38, 56]. Such systems pose new challenges for the development and application of autonomic computing techniques, due to their special characteristics including *nondeterminism*, *context-awareness*, and *goal- and inference-driven adaptability* [53].

5 Some Categorical Terms

In this section, we recall some concepts from the category theory [2, 6, 7, 29, 30] used in this chapter.

5.1 What is a category?

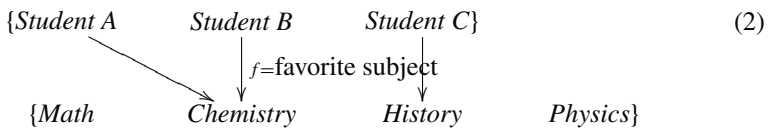
■ A category \mathbf{C} can be viewed as a graph $(Obj(\mathbf{C}), Arc(\mathbf{C}), s, t)$, where

- $Obj(\mathbf{C})$ is the set of nodes we call *objects*,
- $Arc(\mathbf{C})$ is the set of edges we call *morphisms* and
- $s, t : Arc(\mathbf{C}) \rightarrow Obj(\mathbf{C})$ are two maps called *source* (or *domain*) and *target* (or *codomain*), respectively.

We write $f : \mathcal{X} \rightarrow \mathcal{Y}$ when f is in $Arc(\mathbf{C})$ and $s(f) = \mathcal{X}$ and $t(f) = \mathcal{Y}$.

Explanation on terminology: An object in the category is an algebraic structure such as a set. We are probably familiar with some notations for finite sets: $\{Student A, Student B, Student C\}$ is a name for the set whose three elements are *Student A*, *Student B*, *Student C*. Note that the order in which the elements are listed is irrelevant.

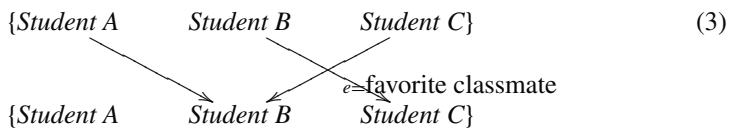
A morphism f in the category consists of three things: a set \mathcal{X} , called the source of the morphism; a set \mathcal{Y} , called the target of the morphism; and a rule assigning to each element x in the source an element y in the target. This y is denoted by $f(x)$, read “ f of x .” Note that the morphism is also called the *map*, *function*, *transformation*, *operator*, or *arrow*. For example, let $\mathcal{X} = \{Student A, Student B, Student C\}$, $\mathcal{Y} = \{Math, Physics, Chemistry, History\}$ and let f assign each student his or her favorite subject. The following internal diagram is an illustration.



This states that the favorite subject of the *Student C* is *History*, written by $f(Student C) = History$, while *Student A* and *Student B* prefer *Chemistry*. There are some important properties of any morphism

- From each element in the source $\{Student A, Student B, Student C\}$, there is exactly one arrow leaving.
- To an element in the target $\{Math, Physics, Chemistry, History\}$, there may be zero, one or more arrows arriving.

It is possible that the source and target of the morphism could be the same set. The following internal diagram is an example.



and, in the case, the morphism is called an *endomorphism* whose representation is available as in



■ Associated with each object \mathcal{X} in $Obj(\mathbf{C})$, there is a morphism $1_{\mathcal{X}} = \mathcal{X} \longrightarrow \mathcal{X}$, called the *identity* morphism on \mathcal{X} , and to each pair of morphisms $f : \mathcal{X} \longrightarrow \mathcal{Y}$ and $g : \mathcal{Y} \longrightarrow \mathcal{Z}$, there is an associated morphism $f;g : \mathcal{X} \longrightarrow \mathcal{Z}$, called the *composition* of f with g . The representations in (5) include the external diagrams of identity morphism and composition of morphisms.

$$\begin{array}{c}
 \begin{array}{c} \curvearrowright \\ \mathcal{X} \end{array} \\
 \mathcal{X} \xrightarrow{f} \mathcal{Y} \xrightarrow{g} \mathcal{Z} \\
 \underbrace{\hspace{10em}}_{f;g}
 \end{array} \tag{5}$$

Explanation on terminology: Here are the corresponding internal diagrams of the identity morphism.

$$\begin{array}{ccc}
 \{Student A & Student B & Student C\} \\
 \downarrow & \downarrow 1_{\mathcal{X}} & \downarrow \\
 \{Student A & Student B & Student C\}
 \end{array} \tag{6}$$

Or

$$\begin{array}{ccc}
 \begin{array}{c} \curvearrowright \\ \{Student A \end{array} & Student B & \begin{array}{c} \curvearrowright \\ Student C \end{array} \\
 & \begin{array}{c} \curvearrowright \\ Student B \end{array} &
 \end{array} \tag{7}$$

And here, the composition of morphisms is described in the internal diagram

$$\begin{array}{ccc}
 \{Student A & Student B & Student C\} \\
 \swarrow & \searrow & \swarrow \\
 \{Student A & Student B & Student C\} \\
 \swarrow & \downarrow f = \text{favorite subject} & \downarrow e = \text{favorite classmate} \\
 \{Math & Chemistry & History & Physics\}
 \end{array} \tag{8}$$

Or, in the external diagram $\mathcal{X} \xrightarrow{e} \mathcal{X} \xrightarrow{f} \mathcal{Y}$. By diagram (8), we can obtain answers for the question “What should each student support to his or her favorite classmate for subject?” In fact, the answers are such as “*Student A* likes *Student B*, *Student B* likes *Chemistry*, so *Student A* should support *Chemistry*,” “*Student B* likes *Student C*, *Student C* likes *History*, so *Student B* should support *History*” and “*Student C* likes *Student B*, *Student B* likes *Chemistry*, so *Student C* should support *Chemistry*.”

The composition of two morphisms e and f means that e and f are combined to obtain a third morphism $\mathcal{X} \xrightarrow{e;f} \mathcal{Y}$. This is represented in the following internal diagram.

$$\begin{array}{ccc}
 \{Student A & Student B & Student C\} \\
 \swarrow & \searrow & \swarrow \\
 \{Math & Chemistry & History & Physics\}
 \end{array} \tag{9}$$

where, for example, $e; f(Student B) = History$ is read as “the favorite subject of the favorite classmate of *Student B* is *History*.”

■ The following equation must hold for all objects \mathcal{X}, \mathcal{Y} in $Obj(\mathbf{C})$ and morphism $f : \mathcal{X} \rightarrow \mathcal{Y}$ in $Arc(\mathbf{C})$:

$$\begin{aligned}
 & \text{Identity:} \quad 1_{\mathcal{X}}; f = f = f; 1_{\mathcal{Y}} \tag{10} \\
 & 1_{\mathcal{X}} \left(\curvearrowright \mathcal{X} \xrightarrow{f} \mathcal{Y} = \mathcal{X} \xrightarrow{f} \mathcal{Y} = \mathcal{X} \xrightarrow{f} \mathcal{Y} \left(\curvearrowright 1_{\mathcal{Y}}
 \end{aligned}$$

The following equation must hold for all objects \mathcal{X}, \mathcal{Y} and \mathcal{Z} in $Obj(\mathbf{C})$ and morphisms $f : \mathcal{X} \rightarrow \mathcal{Y}, g : \mathcal{Y} \rightarrow \mathcal{Z}$ and $h : \mathcal{Z} \rightarrow \mathcal{T}$ in $Arc(\mathbf{C})$:

$$\begin{aligned}
 & \text{Associativity:} \quad (f; g); h = f; (g; h) \tag{11} \\
 & \underbrace{\mathcal{X} \xrightarrow{f} \mathcal{Y} \xrightarrow{g} \mathcal{Z} \xrightarrow{h} \mathcal{T}}_{f;g} = \mathcal{X} \xrightarrow{f} \mathcal{Y} \xrightarrow{g} \mathcal{Z} \xrightarrow{h} \mathcal{T} \underbrace{\hspace{10em}}_{g;h}
 \end{aligned}$$

5.2 Isomorphism

A morphism $f : \mathcal{X} \rightarrow \mathcal{Y}$ in the category \mathbf{C} is an *isomorphism* if there exists a morphism $g : \mathcal{Y} \rightarrow \mathcal{X}$ in that category such that $f; g = 1_{\mathcal{X}}$ and $g; f = 1_{\mathcal{Y}}$.

$$\begin{aligned}
 & \underbrace{\mathcal{X} \xrightarrow{f} \mathcal{Y} \xrightarrow{g} \mathcal{X}}_{f;g=1_{\mathcal{X}}} \quad \text{and} \quad \underbrace{\mathcal{Y} \xrightarrow{g} \mathcal{X} \xrightarrow{f} \mathcal{Y}}_{g;f=1_{\mathcal{Y}}} \tag{12}
 \end{aligned}$$

That is, if the following diagram commutes.

$$\begin{aligned}
 & \begin{array}{ccc}
 & & \mathcal{Y} \\
 & \swarrow g & \\
 1_{\mathcal{X}} \left(\curvearrowright \mathcal{X} & & \mathcal{Y} \left(\curvearrowright 1_{\mathcal{Y}} \right. \\
 & \searrow f & \\
 & & \mathcal{X}
 \end{array} \tag{13}
 \end{aligned}$$

5.3 Element of a set

For any set $A, x \in A$ iff $1 \xrightarrow{x} A$ (or $x : 1 \rightarrow A$) where 1 denotes a singleton set. Focus on one element of $\{Math, Physics, Chemistry, History\}$, say $\{subject\}$, and call this set “1.” Let us see what the morphisms from 1 to $\{Math, Physics, Chemistry, History\}$ are. There are exactly four of them.

$$\{subject\} \quad \{Math \quad Chemistry \quad History \quad Physics\} \quad (14)$$

$\underbrace{\hspace{10em}}_{Math}$

$$\{subject\} \quad \{Math \quad Chemistry \quad History \quad Physics\} \quad (15)$$

$\underbrace{\hspace{10em}}_{Chemistry}$

$$\{subject\} \quad \{Math \quad Chemistry \quad History \quad Physics\} \quad (16)$$

$\underbrace{\hspace{10em}}_{History}$

$$\{subject\} \quad \{Math \quad Chemistry \quad History \quad Physics\} \quad (17)$$

$\underbrace{\hspace{10em}}_{Physics}$

By this way, we can write $1 \xrightarrow{2} \mathbb{N}$ (or $2 : 1 \longrightarrow \mathbb{N}$) for $2 \in \mathbb{N}$, $1 \xrightarrow{i} \mathbb{N}$ (or $i : 1 \longrightarrow \mathbb{N}$) for $i \in \mathbb{N}$ and so on.

5.4 Functor

Functor is a special type of mapping between categories. Functor from a category to itself is called an *endofunctor*. Note that the functors are also viewed as morphisms in a category, whose objects are smaller categories.

5.5 T-algebra

Let \mathbf{C} be a category, \mathcal{A} an object in $Obj(\mathbf{C})$, $T : \mathbf{C} \longrightarrow \mathbf{C}$ an endofunctor and f a morphism $T(\mathcal{A}) \xrightarrow{f} \mathcal{A}$; then *T-algebra* is a pair $\langle \mathcal{A}, f \rangle$. $Obj(\mathbf{C})$ is called a *carrier* of the algebra and T a *signature* of the algebra.

6 Self-* in ANCSs

As known that ANC, and thus self-*, is achieved when ANCSs are constructed. In this way, for forming ANCSs, we start with considering *deterministic autonomic networked computing systems* (DANCSs) and then extend to *nondeterministic autonomic networked computing systems* (NANCSs) by categorical approach in this section.

6.1 Self-* in DANCSs

DANC we want to abstract is intuitionally multiple partial morphism applications, such as

$$s_0 \xrightarrow{\sigma_0} s_1 \xrightarrow{\sigma_1} s_2 \xrightarrow{\sigma_2} s_3 \cdots \tag{18}$$

where

- All indexes $i \in T (= \mathbb{N} \cup \{0\})$ refer to times,
- s is a state of DANCS in the set, denoted by Sys , of states. s_i is the state s at the time i ,
- σ is a contextual data in the set, denoted by $Context$, of contextual data. σ_i is the contextual data σ at the time i , which makes change of the state s_i to become s_{i+1} .

The meaning of (18) is understood as

$$\dots s_2(s_1(s_0())) = \dots s_2(s_1(\sigma_0)) = \dots s_2(\sigma_1) = \dots \sigma_2 \tag{19}$$

The adaptation process in (18) can also be descriptively drawn as

$$s_0() \ \sigma_0 \ \sigma_1 \ \sigma_2 \ \cdots \dashrightarrow s_1(\sigma_0) \ \sigma_1 \ \sigma_2 \ \cdots \dashrightarrow \sigma_0 \ s_2(\sigma_1) \ \sigma_2 \ \cdots \tag{20}$$

or, in another representation

$$\xrightarrow{s_0} \sigma_0 \ \sigma_1 \ \sigma_2 \ \cdots \dashrightarrow \sigma_0 \xrightarrow{s_1} \sigma_1 \ \sigma_2 \ \cdots \dashrightarrow \sigma_0 \ \sigma_1 \xrightarrow{s_2} \sigma_2 \ \cdots \tag{21}$$

Note that in (20) and (21), we want to represent the above-mentioned adaptation process of DANCS based on context where each step of the process is an application of unary partial morphism $1 \xrightarrow{s_i} Sys$ on $1 \xrightarrow{\sigma_{i-1}} Context$, for all i in T .

The adaptation process, in (20) and (21), describes the notion of DANC in DANCSs including the adaptation steps to change *configurations* of the system.

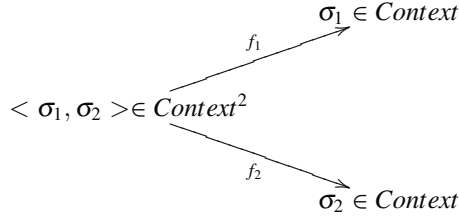
Definition 1 (Configuration of DANCS) *We define a configuration of DANCS at an adaptation step to be a member of the set $Sys \times Context^{i \in T}$, where $Context^{i \in T}$ stands for*

$$Context^{i \in T} = \underbrace{Context \times Context \times \dots \times Context}_{i \text{ times}} \tag{22}$$

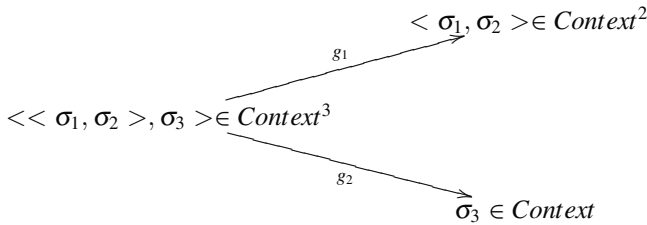
Explanation on terminology: As we know, when we combine sets by multiplication, each set is a *factor* and the resulting set is the *product*. Hence, each set $Context$ is a factor of the resulting set $Context^{i \in T}$, Sys and $Context^{i \in T}$ are two factors of the

set $Sys \times Context^{i \in T}$. The definition of multiplication of sets is very natural. Just remember that a product is not just a set, but a set with two morphisms as in

- When $i = 2$ then $Context^2 = \{ \langle \sigma_1, \sigma_2 \rangle \mid \sigma_1, \sigma_2 \in Context \}$ is obtained by



- When $i = 3$ then $Context^3 = \{ \langle \langle \sigma_1, \sigma_2 \rangle, \sigma_3 \rangle \mid \sigma_1, \sigma_2, \sigma_3 \in Context \}$ is obtained by



Specially, we have

- If $i = 0$ then $Context^0 = \{ \}$
- If $i = 1$ then $Context^1 = Context = \{ \sigma_1 \mid \sigma_1 \in Context \}$

We hope that these diagrams seem suggestive to readers. Our aim is to learn to use them as precise tools of understanding and reasoning, not merely as intuitive guides.

The DANC paradigm, which we want to approach to, is based on mapping a configuration to another. Let us see the following examples

Example 1 A specific DANC can be specified by the following morphism:

$$Self-X : (Sys \times Context) \longrightarrow Sys \tag{23}$$

(i.e., $Self-X : (Sys \times Context^1) \longrightarrow (Sys \times Context^0)$ or denoted by $Self-X (Sys \times Context, Sys)$)

Example 2 Another specific DANC can be specified by

$$Self-X : (Sys \times Context) \longrightarrow (Sys \times Context) \tag{24}$$

(i.e., $Self-X : (Sys \times Context^1) \longrightarrow (Sys \times Context^1)$ or denoted by $Self-X (Sys \times Context, Sys \times Context)$)

Example 3 Again, we can also specify another specific DANC as

$$Self-X : (Sys \times Context^n) \longrightarrow (Sys \times Context) \quad (25)$$

(i.e., $Self-X : (Sys \times Context^n) \longrightarrow (Sys \times Context^1)$ or denoted by $Self-X (Sys \times Context^n, Sys \times Context)$)

and we can, in the completely same way, do for any other specific DANC.

Definition 2 Generally, an arbitrary DANC is specified by

$$Self-X : (Sys \times Context^{i \in T}) \longrightarrow (Sys \times Context^{j \in T}) \quad (26)$$

Now, let us try to do the following exercise.

Exercise 1 (Self-* in DANCs) Show that the morphism $Self-X$ in (26) defines self-* in DANCs

Solution This stems from (26) and the fact that ANC, and thus DANC, is described through self*.
□

Morphism $Self-X$ is called a self-*. Morphism $Self-X$ in (26) defines a set $\{Self-X_{k \in \mathbb{N}}\}$ of mappings such that

$$\{Self-X_{k \in \mathbb{N}}\} : (Sys \times Context^{i \in T}) \longrightarrow (Sys \times Context^{j \in T}) \quad (27)$$

Hence, let us do the exercise as in

Exercise 2 (Self-* facets in DANCs) Show that the set $\{Self-X_{k \in \mathbb{N}}\}$ in (27) defines self-* facets in DANCs. Each mapping $Self-X_{k \in \mathbb{N}}$ is called a self-* facet.

Solution This originates as the result of the truth that self-* is the set of self-* facets.
□

For further well-founded investigation, we can construct a category of the sets of DANC configurations and establish $Self-X$ -algebras as described in the following exercises.

Exercise 3 (Category of the sets of DANC configurations) Show that the sets of DANC configurations as in Definition 1 define a category.

Solution In fact, let $\mathbf{Cat}(\mathbf{DANCs})$ be such a category of the sets of DANC configurations, whose structure is constructed as follows:

- Each set of configurations $Sys \times Context^{i \in T}$ defines an object.
That is, $Obj(\mathbf{Cat}(\mathbf{DANCs})) = \{Sys \times Context^{i \in T}\}$.
- Each $Self-X$ defines a morphism.
That is, $Arc(\mathbf{Cat}(\mathbf{DANCs})) = \{Self-X : (Sys \times Context^{i \in T}) \longrightarrow (Sys \times Context^{j \in T})\}$.

It is easy to check that identity in (10) and associativity in (11) on all *Self-X*s are satisfied. □

Exercise 4 (*Self-X-algebra(DANCS)*) Show that each morphism *Self-X* in the category **Cat(DANCS)** defines an algebra, so-called *Self-X-algebra (DANCS)*.

Solution This stems from definition of T-algebra in Section 5, where functor T is defined such that $T = \biguplus\{Self-X\}$. Note that the notation \biguplus stands for *disjoint union* or *coproduct*. □

With the result of Exercise 4, we obtain a compact formal definition of DANCS as in

Definition 3 (DANCS) *Each Self-X-algebra(DANCS) defines a DANCS*

Both *Sys* and *Context* may be infinite. If both *Sys* and *Context* are finite, then we have a finite DANCS, otherwise we have an infinite DANCS.

6.2 *Self-** in NANCs

In NANC we want to model is intuitively multiple partial morphism applications, such as

$$s_0 \xrightarrow{\sigma_0|x_0} s_1 \xrightarrow{\sigma_1|x_1} s_2 \xrightarrow{\sigma_2|x_2} s_3 \dots \tag{28}$$

where

- All indexes *i* in *T*, *s_i*, and σ_i are similar in meaning to the ones mentioned in (18)
- *x_i* is a real number that can be thought of as the *multiplicity* (or *weight*) with which the adaptation from *s_i* to *s_{i+1}* occurs.

Adaptation process of NANC in diagram (28) can be separated into two complementary parts as follows:

$$s_0 \xrightarrow{\sigma_0} s_1 \xrightarrow{\sigma_1} s_2 \xrightarrow{\sigma_2} s_3 \dots \tag{29}$$

and

$$s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} s_3 \dots \tag{30}$$

On the one hand, diagram (29) emphasizes $1 \xrightarrow{\sigma_i} Context$, for all *i* in *T*, in the adaptation process. This allows us to discover conveniently sequence of σ_i as series of contextual data. On the other hand, diagram (30) gives rise to $1 \xrightarrow{x_i} \mathbb{R}$, for all *i* in *T*, as weights of the series of contextual data in the adaptation process to support an evaluation of weight-based quantitative behaviors of the series of contextual data. Some first steps of the adaptation process in (28) can also be descriptively drawn as

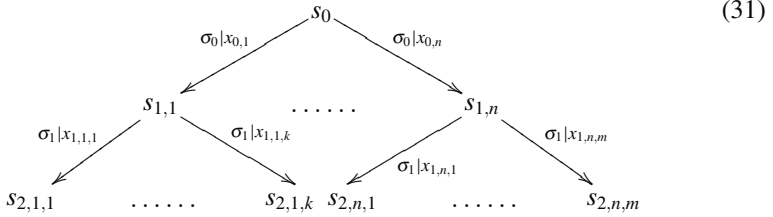


Diagram (31) is thought of as

- For the first step,

$$s_1 \in \{s_{1,1}, \dots, s_{1,n}\} \subset Sys$$

and

$$x_0 \in \{x_{0,1}, \dots, x_{0,n}\} \subset \mathbb{R}$$

- For the second step,

$$s_2 \in \{s_{2,1,1}, \dots, s_{2,1,k}\} \cup \dots \cup \{s_{2,n,1}, \dots, s_{2,n,m}\} \subset Sys$$

and

$$x_1 \in \{x_{1,1,1}, \dots, x_{1,1,k}\} \cup \dots \cup \{x_{1,n,1}, \dots, x_{1,n,m}\} \subset \mathbb{R}$$

and the meaning of (28) is viewed as the following morphism.

$$Self-X : (Sys \times Context) \longrightarrow (Sys \longrightarrow \mathbb{R}) \tag{32}$$

Explanation on terminology: The adaptation morphism *Self-X* in (32) is nondeterministic and this can be explained as follows: *Self-X* assigns to each configuration in $Sys \times Context$ a morphism $Sys \longrightarrow \mathbb{R}$ that can be seen as a kind of *nondeterministic configuration* (or so-called *distributed configuration*) and specifies for every state s' in Sys a multiplicity (or weight) $Self-X(\langle s, \sigma \rangle)(s')$ in \mathbb{R} .

This nondeterminism of NANC makes extension in representation of the categorical models mentioned in Section 6.1. Let us see the following examples

Example 4 A specific NANC, which is specified by the following morphism, is an extension of (23):

$$Self-X : (Sys \times Context) \longrightarrow (Sys \longrightarrow \mathbb{R}) \tag{33}$$

(i.e., $Self-X : (Sys \times Context^1) \longrightarrow ((Sys \times Context^0) \longrightarrow \mathbb{R})$ or denoted by $Self-X((Sys \times Context), (Sys \longrightarrow \mathbb{R}))$)

Example 5 The model in (24) extended for NANC is specified by

$$Self-X : (Sys \times Context) \longrightarrow ((Sys \times Context) \longrightarrow \mathbb{R}) \tag{34}$$

(i.e., $Self-X : (Sys \times Context^1) \longrightarrow ((Sys \times Context^1) \longrightarrow \mathbb{R})$ or denoted by $Self-X((Sys \times Context), ((Sys \times Context) \longrightarrow \mathbb{R}))$)

Example 6 Again, we specify another specific NANC as an extension of (25) in

$$Self-X : (Sys \times Context^n) \longrightarrow ((Sys \times Context) \longrightarrow \mathbb{R}) \quad (35)$$

(i.e., $Self-X : (Sys \times Context^n) \longrightarrow ((Sys \times Context^1) \longrightarrow \mathbb{R})$ or denoted by $Self-X ((Sys \times Context^n), ((Sys \times Context) \longrightarrow \mathbb{R}))$)

and, in the completely same way, we do for an arbitrary NANC as in

Definition 4 Generally, an arbitrary NANC is specified by

$$Self-X : (Sys \times Context^{i \in T}) \longrightarrow ((Sys \times Context^{j \in T}) \longrightarrow \mathbb{R}) \quad (36)$$

Let us do the exercise as described in

Exercise 5 (Self-* in NANCSS) Show that the morphism $Self-X$ in (36) defines self-* in NANCSS

Solution This stems from (36) and the fact that ANC, and thus NANC, is described through self-*. □

Morphism $Self-X$ in (36) defines a set $\{Self-X_{k \in \mathbb{N}}\}$ of mappings such that

$$\{Self-X_{k \in \mathbb{N}}\} : (Sys \times Context^{i \in T}) \longrightarrow ((Sys \times Context^{j \in T}) \longrightarrow \mathbb{R}) \quad (37)$$

Thus, let us do the following exercises

Exercise 6 (Self-* facets in NANCSS) Show that the set $\{Self-X_{k \in \mathbb{N}}\}$ in (37) defines self-* facets in NANCSS. Each mapping $Self-X_{k \in \mathbb{N}}$ is called a self-* facet.

Solution This originates as the result of the truth that self-* is the set of self-* facets. □

Exercise 7 (Category of the sets of NANC configurations) Show that the category $\mathbf{Cat}(\mathbf{DANC})$ equipped with structure $(Sys \times Context^{i \in T}) \longrightarrow ((Sys \times Context^{j \in T}) \longrightarrow \mathbb{R})$ defines a category $\mathbf{Cat}(\mathbf{NANC})$ of the sets of NANC configurations.

Solution This result comes immediately from Exercise 3. □

Exercise 8 (Self-X-algebra(NANC)) Show that the structure $(Sys \times Context^{i \in T}) \longrightarrow ((Sys \times Context^{j \in T}) \longrightarrow \mathbb{R})$ in the category $\mathbf{Cat}(\mathbf{NANC})$ defines an algebra, so-called $Self-X$ -algebra (NANC).

Solution This originates from definition on T-algebra in Section 5, where functor T is defined such that $T = \biguplus\{Self-X\}$ (similar to Exercise 4) with $Self-X$ defined in (36). □

With this result of Exercise 8, we obtain a compact formal definition of NANC as in

Definition 5 (NANCS) Each Self- X -algebra (NANCS) defines a NANCS

Moreover, let us do the following exercise to obtain a significant relationship between DANCSs and NANCSs.

Exercise 9 (Relationship between DANCSs and NANCSs) Show that DANCSs are just of specific NANCSs. In other words, using categorical language, DANCSs $\xrightarrow{c} \text{NANCSs}$

Solution In fact, by the adaptation morphism in (36) of NANCSs, let f be the morphism $f : (\text{Sys} \times \text{Context}^{j \in T}) \longrightarrow \mathbb{R}$, Conf be $\text{Sys} \times \text{Context}^{j \in T}$ and the finite set $\mathbb{R}(\text{Conf}) = \{1 \xrightarrow{c} \text{Conf} \mid f(c) \neq 0\} \xrightarrow{\subseteq} \text{Conf}$. Hence it follows that when $\exists! 1 \xrightarrow{c} \text{Conf} : f(c) = 1$ but $\forall c' \neq c : f(c') = 0$ (i.e., the set $\mathbb{R}(\text{Conf})$ is a singleton set of configuration with weight of 1. Note that the notation $\exists!$ is read as “exist only”) then (36) becomes the adaptation morphism of DANCSs as in (26). In other words, in the case, NANCSs will become DANCSs. \square

7 Structures of Self-*

In this section, we construct self-* monoid and then a category of self-* monoids in order to consider the significant properties of the self-*.

7.1 Self-* Monoid

We know that self-* is specified by the morphism $\text{Self-}X : (\text{Sys} \times \text{Context}^{n \in T}) \longrightarrow (\text{Sys} \times \text{Context}^{n \in T})$, which defines the set $\{\text{Self-}X_{i \in \mathbb{N}}(\text{Sys} \times \text{Context}^{n \in T}, \text{Sys} \times \text{Context}^{n \in T})\}$ of self-* facets. Let $\mathbf{Self-X}^{n \in T}$ be the set of such self-* facets, then

$$\mathbf{Self-X}^{n \in T} = \{\text{Self-}X_{i \in \mathbb{N}}(\text{Sys} \times \text{Context}^{n \in T}, \text{Sys} \times \text{Context}^{n \in T})\} \quad (38)$$

Note that, in the case, we write $\text{Self-}X_{i \in \mathbb{N}}^{n \in T}$ to stand for $\text{Self-}X_{i \in \mathbb{N}}(\text{Sys} \times \text{Context}^{n \in T}, \text{Sys} \times \text{Context}^{n \in T})$. Thus, we have

$$\mathbf{Self-X}^{n \in T} = \{\text{Self-}X_{i \in \mathbb{N}}^{n \in T}\} \quad (39)$$

This set with the composition operation “;” satisfies two following properties.

7.1.1 Composition of Self-* Facets

Let f and g be members of $\mathbf{Self-X}^{n \in T}$, then the composition of self-* facets $f;g : (\text{Sys} \times \text{Context}^{n \in T}) \longrightarrow (\text{Sys} \times \text{Context}^{n \in T})$ is as $g : (f : (\text{Sys} \times \text{Context}^{n \in T}))$

$\longrightarrow (\text{Sys} \times \text{Context}^{n \in T}) \longrightarrow (\text{Sys} \times \text{Context}^{n \in T})$. In other words, let $f = \text{Self-X}_{i \in \mathbb{N}}^{n \in T}$ and $g = \text{Self-X}_{j \in \mathbb{N}}^{n \in T}$ then

$$(\text{Self-X}_{i \in \mathbb{N}}^{n \in T} ; \text{Self-X}_{j \in \mathbb{N}}^{n \in T}) = \text{Self-X}_{j \in \mathbb{N}}(\text{Self-X}_{i \in \mathbb{N}}^{n \in T}, \text{Sys} \times \text{Context}^{n \in T}) \quad (40)$$

7.1.2 Identity of Self-* Facets

There exist identities $1_{n \in T} : (\text{Sys} \times \text{Context}^{n \in T}) \longrightarrow (\text{Sys} \times \text{Context}^{n \in T})$ of self-* facets in $\mathbf{Self-X}^{n \in T}$ such that, for every f in $\mathbf{Self-X}^{n \in T}$, $1_{n \in T}; f = f; 1_{n \in T} = f$ to be held. In other words, this can be specified by

$$\begin{aligned} \text{Self-X}_{i \in \mathbb{N}}^{n \in T} &= \text{Self-X}_{i \in \mathbb{N}}(1_{n \in T}, \text{Sys} \times \text{Context}^{n \in T}) \\ &= \text{Self-X}_{i \in \mathbb{N}}(\text{Sys} \times \text{Context}^{n \in T}, 1_{n \in T}) \\ &= \text{Self-X}_{i \in \mathbb{N}}(\text{Sys} \times \text{Context}^{n \in T}, \text{Sys} \times \text{Context}^{n \in T}) \end{aligned} \quad (41)$$

Thus, $\mathbf{Self-X}^{n \in T}$ with the composition operation “;” is called *self-* monoid*. Moreover, the monoid $\mathbf{Self-X}^{n \in T}$ is also a monoid category including only one object to be the set $\{\text{Self-X}_{i \in \mathbb{N}}^{n \in T}\}$, each of whose members is a self-* facet, and by the composition operation as a morphism, then the associativity and identity on the morphisms are completely satisfied.

7.2 A Category of Self-* Monoids

By the self-* monoids $\mathbf{Self-X}^{i \in T}$, we can construct $\mathbf{Cat}(\mathbf{Self-X})$ to be a category of self-* monoids. In fact, $\mathbf{Cat}(\mathbf{Self-X})$ is constructed as follows:

- *Objects:* $\text{Obj}(\mathbf{Cat}(\mathbf{Self-X}))$ is the set of self-* monoids $\mathbf{Self-X}^{i \in T}$. That is,

$$\text{Obj}(\mathbf{Cat}(\mathbf{Self-X})) = \{\mathbf{Self-X}^{i \in T}\} \quad (42)$$

- *Morphisms:* Associated with each object $\mathbf{Self-X}^{i \in T}$ in $\text{Obj}(\mathbf{Cat}(\mathbf{Self-X}))$, we define a morphism $\mathbf{Self-X}^{i \in T} \xrightarrow{1_{\mathbf{Self-X}^{i \in T}}} \mathbf{Self-X}^{i \in T}$, the identity morphism on $\mathbf{Self-X}^{i \in T}$ such that

$$\mathbf{Self-X}^{i \in T} \xrightarrow{1_{\mathbf{Self-X}^{i \in T}} \stackrel{\text{def}}{=} 1_{i \in T}} \mathbf{Self-X}^{i \in T} \quad (43)$$

or

$$\{\text{Self-X}_{k \in \mathbb{N}}^{i \in T}\} \xrightarrow{1_{\mathbf{Self-X}^{i \in T}} \stackrel{\text{def}}{=} 1_{i \in T}} \{\text{Self-X}_{k \in \mathbb{N}}^{i \in T}\} \quad (44)$$

and to each pair of morphisms $\mathbf{Self-X}^{i \in T} \xrightarrow{f} \mathbf{Self-X}^{j \in T}$ and $\mathbf{Self-X}^{j \in T} \xrightarrow{g} \mathbf{Self-X}^{i \in T}$ such that

$$\mathbf{Self-X}^{i \in T} \xrightarrow{f \stackrel{def}{=} 1_{i \in T} \times \text{Context}^{j-i}} \mathbf{Self-X}^{j \in T} \quad (45)$$

and

$$\mathbf{Self-X}^{j \in T} \xrightarrow{g \stackrel{def}{=} 1_{j \in T} \times \text{Context}^{k-j}} \mathbf{Self-X}^{k \in T} \quad (46)$$

there is an associated morphism $\mathbf{Self-X}^{i \in T} \xrightarrow{f;g} \mathbf{Self-X}^{k \in T}$, the composition of f with g , such that

$$\mathbf{Self-X}^{i \in T} \xrightarrow{f;g = 1_{i \in T} \times \text{Context}^{k-i}} \mathbf{Self-X}^{k \in T} \quad (47)$$

For every object in $Obj(\mathbf{Cat}(\mathbf{Self-X}))$ and the morphisms

$$\mathbf{Self-X}^{i \in T} \xrightarrow{f \stackrel{def}{=} 1_{i \in T} \times \text{Context}^{j-i}} \mathbf{Self-X}^{j \in T} \quad (48)$$

$$\mathbf{Self-X}^{j \in T} \xrightarrow{g \stackrel{def}{=} 1_{j \in T} \times \text{Context}^{k-j}} \mathbf{Self-X}^{k \in T} \quad (49)$$

and

$$\mathbf{Self-X}^{k \in T} \xrightarrow{h \stackrel{def}{=} 1_{k \in T} \times \text{Context}^{m-k}} \mathbf{Self-X}^{m \in T} \quad (50)$$

in $Arc(\mathbf{Cat}(\mathbf{Self-X}))$, the following equations hold:

Associativity: $(f; g); h = f; (g; h) = 1_{i \in T} \times \text{Context}^{m-i}$

Identity: $1_{\mathbf{Self-X}^{i \in T}}; f = f = f; 1_{\mathbf{Self-X}^{j \in T}}$

(i.e., $1_{i \in T}; 1_{i \in T} \times \text{Context}^{j-i} = 1_{i \in T} \times \text{Context}^{j-i} = 1_{i \in T} \times \text{Context}^{j-i}; 1_{j \in T}$)

As a result, the above-mentioned monoid morphisms can be diagrammatically drawn such as

$$\mathbf{Self-X}^{i \in T} \xrightarrow{1_{i \in T} \times \text{Context}^{\pm k}} \mathbf{Self-X}_{i \pm k \in T} \quad (51)$$

or

$$\{\mathbf{Self-X}_{i \in \mathbb{N}}^{i \in T}\} \xrightarrow{1_{i \in T} \times \text{Context}^{\pm k}} \{\mathbf{Self-X}_{i \in \mathbb{N}}^{i \pm k \in T}\} \quad (52)$$

These are all the basic ingredients we need to have the category $\mathbf{Cat}(\mathbf{Self-X})$. Let us see a general definition of category presented in Section 5 for reference.

7.3 Some Properties of Category $\mathbf{Cat}(\mathbf{Self-X})$

By the construction of category $\mathbf{Cat}(\mathbf{Self-X})$, some emerging significant properties are presented in this subsection.

Property 1 All monoid morphisms of $\mathbf{Cat}(\mathbf{Self-X})$ are monoid isomorphisms.

Proof This result immediately stems from diagram (51). In fact, for every pair of monoid morphisms in $Arc(\mathbf{Cat}(\mathbf{Self-X}))$ between $\mathbf{Self-X}^{i \in T}$ and $\mathbf{Self-X}^{j \in T}$, we always have the following diagram:

$$\begin{array}{ccc}
 & \xrightarrow{1_{i \in T} \times Context^{j-i}} & \\
 \mathbf{Self-X}^{i \in T} & \xrightarrow{\quad} & \mathbf{Self-X}^{j \in T} \\
 & \xleftarrow{1_{j \in T} \times Context^{i-j}} & \\
 & \text{with self-loops } 1_{i \in T} \text{ and } 1_{j \in T} &
 \end{array} \tag{53}$$

These monoid morphisms satisfy an isomorphic relationship. Q.E.D.

Property 2 Isomorphisms between any pair of monoids in $\mathbf{Cat}(\mathbf{Self-X})$ are ever isomorphisms between the pair of ANCSs.

Proof This comes from the fact that each object of category $\mathbf{Cat}(\mathbf{Self-X})$ is just an ANCS. Q.E.D.

From the above-mentioned justification of $\mathbf{Cat}(\mathbf{Self-X})$, we are able to derive $\mathbf{Self-X}^{i \in T}$. Derivation of every $\mathbf{Self-X}^{i \in T}$ is simplified by the following facts:

Property 3 There exists always a self-* monoid $\mathbf{Self-X}$, as simply as it can, in $\mathbf{Cat}(\mathbf{Self-X})$ constructed. Hence, it is available to start with.

Proof It emerges that

$$\begin{aligned}
 \mathbf{Self-X} &= \{Self-X_{i \in \mathbb{N}}(Sys \times Context^0, Sys \times Context^0)\} \\
 &= \{Self-X_{i \in \mathbb{N}}(Sys, Sys)\}
 \end{aligned} \tag{54}$$

thus

$$1 \xrightarrow{\mathbf{Self-X}} Obj(\mathbf{Cat}(\mathbf{Self-X})) \tag{55}$$

Q.E.D.

Property 4 Given $\mathbf{Self-X}$, we can compute $\mathbf{Self-X}^{i \in T}$.

Proof We evaluate self-* monoid $\mathbf{Self-X}^{i \in T}$ such that

$$1 \xrightarrow{\mathbf{Self-X}^{i \in T}} Obj(\mathbf{Cat}(\mathbf{Self-X})) \tag{56}$$

based on the facts that

$$\left(\begin{array}{c} 1 \xrightarrow{\mathbf{Self-X}} \mathit{Obj}(\mathbf{Cat}(\mathbf{Self-X})) \\ \text{and} \\ \mathbf{Self-X} \xrightarrow{1_0 \times \mathit{Context}^i} \mathbf{Self-X}^{i \in T} \end{array} \right) \quad (57)$$

Note that $\mathbf{Self-X} \xrightarrow{1_0} \mathbf{Self-X}$. Q.E.D.

Property 5 Given $\mathbf{Self-X}^{i \in T}$, we can compute $\mathbf{Self-X}^{j \in T}$ for every $j \neq i$.

Proof Self-* monoid $\mathbf{Self-X}^{j \in T}$ is evaluated such that

$$1 \xrightarrow{\mathbf{Self-X}^{j \in T}} \mathit{Obj}(\mathbf{Cat}(\mathbf{Self-X})) \quad (58)$$

based on the facts that

$$\left(\begin{array}{c} 1 \xrightarrow{\mathbf{Self-X}^{i \in T}} \mathit{Obj}(\mathbf{Cat}(\mathbf{Self-X})) \\ \text{and} \\ \mathbf{Self-X}^{i \in T} \xrightarrow{1_{i \in T} \times \mathit{Context}^{j-i}} \mathbf{Self-X}^{j \in T} \end{array} \right) \quad (59)$$

Q.E.D.

From the construction of $\mathbf{Cat}(\mathbf{Self-X})$, we see that every $\mathbf{Self-X}^{i \in T}$ can be formed in the unifying way based on Properties 3–5. As a result, we gain a substantial procedure of construction at a high abstract level without any excessive inclination toward a specific implementation detail. This is quite helpful when we want to justify whether or not some certain properties of the construction are true. In fact, we can prove

Property 6 Every monoid $\mathbf{Self-X}^{i \in T}$ can be constructed by any other monoid in $\mathbf{Cat}(\mathbf{Self-X})$

Proof Applying Properties 3–5 to construct every monoid $\mathbf{Self-X}^{i \in T}$ from another monoid in $\mathbf{Cat}(\mathbf{Self-X})$. Q.E.D.

This is certainly a property we expect of any construction procedure.

Property 7 $\mathbf{Cat}(\mathbf{Self-X})$ is a complete graph

Proof In fact, this is a consequence stemming from Property 6. Q.E.D.

This is indeed a property of our abstract construction mechanism.

8 Series of Self-* Facets

A number of different notations are in use for denoting series of self-* facets.

$$sf = (f_0, f_1, f_2, \dots) \quad (60)$$

is a common notation which specifies a series of self-* facets sf which is indexed by the natural numbers in $T(= \mathbb{N} \cup \{0\})$. We are also accustomed to

$$sf = (f_{t \in T}) \tag{61}$$

Informally, series of self-* facets can be understood as a rope on which we hang up a sequence of self-* facets for display. Hence it follows that

Definition 6 (Series of self-* facets) For morphisms $1 \xrightarrow{t} T$ and $1 \xrightarrow{f_t} \mathbf{Self-X}^{n \in T}$, there exists a unique morphism $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ such that the equation $t; sf = f_t$ holds. This is described by the following commutative diagram

$$\begin{array}{ccc}
 1 & \xrightarrow{t} & T \\
 & \searrow f_t & \downarrow sf \\
 & & \mathbf{Self-X}^{n \in T}
 \end{array}
 \tag{62}$$

Morphism $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ defines a series of self-* facets.

Explanation on semantics: Note that morphism $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ is read as

$$\forall t [t \in T \implies \exists! f_t [f_t \in \mathbf{Self-X}^{n \in T} \ \& \ sf(t) = f_t]]$$

In other words, $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ generates series of self-* facets as an infinite sequence of $sf(0) = f_0, sf(1) = f_1, \dots, sf(t) = f_t, \dots$ which is written as $(sf(0), sf(1), \dots, sf(t), \dots)$ or $(f_0, f_1, \dots, f_t, \dots)$

Definition 7 (Set of series of self-* facets) Given $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ then the set of series of self-* facets, denoted by $\mathbf{Self-X}_\omega^{n \in T}$, is defined by

$$\mathbf{Self-X}_\omega^{n \in T} = \{sf \mid T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}\} \tag{63}$$

We obtain

Corollary 1 If $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ then $1 \xrightarrow{sf} \mathbf{Self-X}_\omega^{n \in T}$

Proof This result stems immediately from Definitions 6 and 7. Q.E.D.

Explanation on semantics: This corollary means that for each morphism $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$, there is a morphism $1 \xrightarrow{sf} \mathbf{Self-X}_\omega^{n \in T}$ generating member in $\mathbf{Self-X}_\omega^{n \in T}$.

That is, morphism $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ generates series of self-* facets and $1 \xrightarrow{sf} \mathbf{Self-X}_{\omega}^{n \in T}$ constructs the set of series of self-* facets.

For series of self-* facets, we can define a mechanism to generate them. This mechanism consists of an object T equipping with structural morphisms $1 \xrightarrow{0} T \xrightarrow{succ} T$ with the property that for $\mathbf{Self-X}^{n \in T}$, any $1 \xrightarrow{f_0} \mathbf{Self-X}^{n \in T}$ and $\mathbf{Self-X}^{n \in T} \xrightarrow{next} \mathbf{Self-X}^{n \in T}$ then there exists a unique morphism $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ such that the following diagram commutes

$$\begin{array}{ccccc}
 1 & \xrightarrow{0} & T & \xrightarrow{succ} & T \\
 & \searrow f_0 & \downarrow sf & & \downarrow sf \\
 & & \mathbf{Self-X}^{n \in T} & \xrightarrow{next} & \mathbf{Self-X}^{n \in T}
 \end{array} \tag{64}$$

Definition 8 (Construction of series of self-* facets) We define a construction morphism of series of self-* facets, denoted by \ddagger , such that

$$\mathbf{Self-X}^{n \in T} \times [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \xrightarrow{\ddagger} [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \tag{65}$$

Explanation on semantics: This definition means that $\ddagger(A \times B \xrightarrow{f \times g} C \times D) = A \ddagger B \xrightarrow{f \ddagger g} C \ddagger D$. It follows that any series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ can be represented in a format including two parts of *head* and *tail* to be connected by “ \ddagger ” such that

$$T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} \equiv 1 \xrightarrow{0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} \ddagger 1 \xrightarrow{t > 0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} \tag{66}$$

where $1 \xrightarrow{0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} = sf(0)$ and $1 \xrightarrow{t > 0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} = (sf(1), sf(2), \dots)$ to be called head and tail, respectively.

Definition 9 (Head of series of self-* facets) We define a head construction morphism, denoted by $1 \xrightarrow{0} (-)$, such that

$$1 \xrightarrow{0} (-) : [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \longrightarrow \mathbf{Self-X}^{n \in T} \tag{67}$$

Explanation on semantics: This definition states that $\forall(a \dagger s)[(a \dagger s) \in [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \implies \exists! f_0[f_0 \in \mathbf{Self-X}^{n \in T} \ \& \ 1 \xrightarrow{0} (a \dagger s) = a = f_0]$

It follows that $1 \xrightarrow{0} (T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}) \stackrel{equiv}{\equiv} 1 \xrightarrow{0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$.

Definition 10 (Tail of series of self-* facets) We define a tail construction morphism, denoted by $(-)'$, such that

$$(-)' : [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \longrightarrow [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \quad (68)$$

Explanation on semantics: This definition means that $\forall(a \dagger s)[(a \dagger s) \in [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \implies \exists!(f_1, f_2, \dots)[(f_1, f_2, \dots) \in [T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}] \ \& \ (a \dagger s)' = s = (f_1, f_2, \dots)]$

As a convention, $(-)^{(n)}$ denotes applying recursively the $(-)'$ n times. Thus, specifically, $(-)^{(2)}$, $(-)^{(1)}$, and $(-)^{(0)}$ stand for $((-))'$, $(-)'$, and $(-)$, respectively.

It follows that the first member of series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ is given by

$$1 \xrightarrow{0} ((T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})') \stackrel{equiv}{\equiv} 1 \xrightarrow{1} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} \quad (69)$$

and, in general, for every $k \in T$ the k -th member of series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ is provided by

$$1 \xrightarrow{0} ((T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})^{(k)})' \stackrel{equiv}{\equiv} 1 \xrightarrow{k} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T} \quad (70)$$

Series of self-* facets to be an infinite sequence of all $f_{t \in T}$ is viewed and treated as single mathematical entity, so the derivative of series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ is given by $(T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})'$

Now using this notation for derivative of series of self-* facets, we can specify series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ as in

Definition 11 A series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ can be specified by

- Initial value: $1 \xrightarrow{0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ and
- Differential equation: $((T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})^{(n)})' = (T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})^{(n+1)}$

Explanation on semantics: The initial value of $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ is defined as its first element $1 \xrightarrow{0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$, and the derivative of series of self-* facets, denoted by $(T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})'$, is defined by $((T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})^{(n)})' = (T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})^{(n+1)}$, for any integer n in T . In other words, the initial value and derivative equal the head and tail of $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$, respectively. The behavior of a series of self-* facets $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ consists of two aspects: it allows for the observation of its initial value $1 \xrightarrow{0} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$; and it can make an evolution to the new series of self-* facets $(T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})'$, consisting of the original series of self-* facets from which the first element has been removed. The initial value of $(T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})'$, which is $1 \xrightarrow{0} ((T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})') = 1 \xrightarrow{1} T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ can in its turn be observed, but note that we have to move from $T \xrightarrow{sf} \mathbf{Self-X}^{n \in T}$ to $(T \xrightarrow{sf} \mathbf{Self-X}^{n \in T})'$ first in order to do so. Now a behavioral differential equation defines a series of self-* facets by specifying its initial value together with a description of its derivative, which tells us how to continue.

Note: Every member $f_{t \in T}$ in $\mathbf{Self-X}^{n \in T}$ can be considered as a series of self-* facets in the following manner. For every $f_{t \in T}$ in $\mathbf{Self-X}^{n \in T}$, a unique series of self-* facets is defined by morphism f :

$$1 \xrightarrow{\overbrace{f_i \circ \circ \dots}^{(f_i, \circ, \circ, \dots)}} \mathbf{Self-X}^{n \in T} \xrightarrow{f} \mathbf{Self-X}_\omega^{n \in T} \tag{71}$$

such that the equation $f_i; f = (f_i, \circ, \circ, \dots)$ holds, where \circ denotes empty member (or null member) in $\mathbf{Self-X}^{n \in T}$. Thus $(f_i, \circ, \circ, \dots)$ is in $\mathbf{Self-X}_\omega^{n \in T}$.

Definition 12 (Equivalence) For any $T \xrightarrow{sf1} \mathbf{Self-X}^{n \in T}$ and $T \xrightarrow{sf2} \mathbf{Self-X}^{n \in T}$, $sf1 = sf2$ iff $1 \xrightarrow{t} T \xrightarrow{sf1} \mathbf{Self-X}^{n \in T} = 1 \xrightarrow{t} T \xrightarrow{sf2} \mathbf{Self-X}^{n \in T}$ with every t in T .

Definition 13 (Bisimulation) Bisimulation on $\mathbf{Self-X}_\omega^{n \in T}$ is a relation, denoted by \sim , between series of self-* facets $T \xrightarrow{sf1} \mathbf{Self-X}^{n \in T}$ and $T \xrightarrow{sf2} \mathbf{Self-X}^{n \in T}$ such that if $sf1 \sim sf2$ then $1 \xrightarrow{0} (sf1) = 1 \xrightarrow{0} (sf2)$ and $(sf1)' \sim (sf2)'$.

Two series of self-* facets are bisimilar if, regarding their behaviors, each of the series “simulates” the other and vice versa. In other words, each of the series cannot be distinguished from the other by the observation. Let us do the following exercises related to the bisimulation between series of self-* facets.

Exercise 10 Let $sf, sf1$ and $sf2$ be in $\mathbf{Self-X}_\omega^{n \in T}$. Show that if $sf \sim sf1$ and $sf1 \sim sf2$ then $(sf \sim sf1) \circ (sf1 \sim sf2) = sf \sim sf2$, where the symbol \circ denotes a relational composition. For more descriptive notation, we can write this in the form

$$\frac{sf \sim sf1, sf1 \sim sf2}{(sf \sim sf1) \circ (sf1 \sim sf2) = sf \sim sf2} \quad (72)$$

and conversely, if $sf \sim sf2$ then there exists $sf1$ such that $sf \sim sf1$ and $sf1 \sim sf2$. This can be written as

$$\frac{sf \sim sf2}{\exists sf1 : sf \sim sf1 \quad \text{and} \quad sf1 \sim sf2} \quad (73)$$

Solution Proving (72) originates as the result of the truth that the relational composition between two bisimulations $L_1 \subseteq sf \times sf1$ and $L_2 \subseteq sf1 \times sf2$ is a bisimulation obtained by $L_1 \circ L_2 = \{\langle x, y \rangle \mid x L_1 z \text{ and } z L_2 y \text{ for some } z \in sf1\}$, where $x \in sf, z \in sf1$ and $y \in sf2$.

Proving (73) comes from the fact that there are always $sf1 = sf$ or $sf1 = sf2$ as simply as they can. Hence, (73) is always true in general. \square

Exercise 11 Let $sf_i, \forall i \in \mathbb{N}$, be in $\mathbf{Self-X}_\omega^{n \in T}$ and $\bigcup_{i \in \mathbb{N}}$ be union of a family of sets. Show that

$$\frac{sf \sim sf_i \quad \text{with } i \in \mathbb{N}}{\bigcup_{i \in \mathbb{N}} (sf \sim sf_i) = sf \sim \bigcup_{i \in \mathbb{N}} sf_i} \quad (74)$$

and conversely,

$$\frac{sf \sim \bigcup_{i \in \mathbb{N}} sf_i}{\exists i \in \mathbb{N} : sf \sim sf_i} \quad (75)$$

Solution Proving (74) stems straightforwardly from the fact that sf bisimulates sf_i (i.e., $sf \sim sf_i$) then, sf bisimulates each series in $\bigcup_{i \in \mathbb{N}} sf_i$.

Conversely, proving (75) develops as the result of the fact that for each $\langle x, y \rangle \in \bigcup_{i \in \mathbb{N}} (sf \times sf_i)$, there exists $i \in \mathbb{N}$ such that $\langle x, y \rangle \in sf \times sf_i$. In other words, it is formally denoted by $\bigcup_{i \in \mathbb{N}} (sf \times sf_i) = \{\langle x, y \rangle \mid \exists i \in \mathbb{N} : x \in sf \quad \text{and} \quad y \in sf_i\}$, where $x \in sf$ and $y \in sf_i$. \square

The union of all bisimulations between sf and sf_i (i.e., $\bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$) is the greatest bisimulation. The greatest bisimulation is called the *bisimulation equivalence* or *bisimilarity* [23, 42] (again denoted by the notation \sim).

Exercise 12 Check that the bisimilarity \sim on $\bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$ is an equivalence relation.

Solution In fact, a bisimilarity \sim on $\bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$ is a binary relation \sim on $\bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$, which is reflexive, symmetric and transitive. In other words, the following properties hold for \sim

- Reflexivity:

$$\frac{\forall (a \sim b) \in \bigcup_{i \in \mathbb{N}} (sf \sim sf_i)}{(a \sim b) \sim (a \sim b)} \quad (76)$$

- Symmetry: $\forall (a \sim b), (c \sim d) \in \bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$,

$$\frac{(a \sim b) \sim (c \sim d)}{(c \sim d) \sim (a \sim b)} \quad (77)$$

- Transitivity: $\forall (a \sim b), (c \sim d), (e \sim f) \in \bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$,

$$\frac{((a \sim b) \sim (c \sim d)) \wedge ((c \sim d) \sim (e \sim f))}{(a \sim b) \sim (e \sim f)} \quad (78)$$

to be an equivalence relation on $\bigcup_{i \in \mathbb{N}} (sf \sim sf_i)$. \square

For some constraint α , if $sf1 \sim sf2$ then two series $sf1$ and $sf2$ have the following relation.

$$\frac{sf1 \models \alpha}{sf2 \models \alpha} \quad (79)$$

That is, if series $sf1$ satisfies constraint α then this constraint is still preserved on series $sf2$. Thus it is read as $sf1 \sim sf2$ in the constraint of α (and denoted by $sf1 \sim_{\alpha} sf2$).

For validating whether $sf1 = sf2$, a powerful method is so-called *proof principle of coinduction* [43] that states as follows:

Theorem 1 (Coinduction) For any $T \xrightarrow{sf1} \mathbf{Self-X}^{n \in T}$ and $T \xrightarrow{sf2} \mathbf{Self-X}^{n \in T}$, if $sf1 \sim sf2$ then $sf1 = sf2$.

Proof In fact, for two series of self-* facets $sf1$ and $sf2$ and a bisimulation $sf1 \sim sf2$. We see that by inductive bisimulation for $k \in T$, then $sf1^{(k)} \sim sf2^{(k)}$.

Therefore, by Definition 13, $1 \xrightarrow{0} (sf1^{(k)}) = 1 \xrightarrow{0} (sf2^{(k)})$. By the equivalence

in (70), then $1 \xrightarrow{k} sf1 = 1 \xrightarrow{k} sf2$ with every $k \in T$. It follows that, by Definition 12, we obtain $sf1 = sf2$. Q.E.D.

Hence in order to prove the equivalence between two series of self-* facets $sf1$ and $sf2$, it is sufficient to establish the existence of a bisimulation relation $sf1 \sim sf2$. In other words, using coinduction we can justify the equivalence between two series of self-* facets $sf1$ and $sf2$ in $\mathbf{Self-X}_{\omega}^{n \in T}$.

Exercise 13 (Generating series of self-* facets) For every sf in $\mathbf{Self-X}_{\omega}^{n \in T}$, show that

$$sf = 1 \xrightarrow{0} (sf) \ddagger (sf)' \quad (80)$$

Solution This stems from the coinductive proof principle in Theorem 1. In fact, it is easy to check the following bisimulation $sf \sim 1 \xrightarrow{0} (sf) \ddagger (sf)'$. It follows that $sf = 1 \xrightarrow{0} (sf) \ddagger (sf)'$ □

In (80), operation \ddagger as a kind of series integration, the exercise states that series derivation and series integration are inverse operations. It gives a way to obtain sf from $(sf)'$ and the initial value $1 \xrightarrow{0} (sf)$. As a result, the exercise allows us to reach solution of differential equations in an algebraic manner.

9 Discussions and Comparisons

The aim of this chapter has been both to give an in-depth analysis as well as to present the new material on the notion of self-* computing. Below we briefly discuss the Wang's approach in [53] and compare it with our development.

- In the paper entitled "Toward Theoretical Foundations of Autonomic Computing" [53], a particular way of considering AC has been approached as a novel computing system at the highest level of machine intelligence, whose goal- and inference-driven computational behaviors have been expressed on top of imperative computing (IC) techniques with event-, time-, and interrupt-driven computational behaviors. By that approach, Wang has developed the overarching foundations and engineering paradigms of AC including the notions of behaviorism, cognitive informatics, denotational mathematics, and intelligent science. In particular, the paper has presented the theorems of the necessary and sufficient conditions of IC and AC, and the generic intelligence model of natural and machine intelligence for further dealing with advanced AC techniques and their engineering applications.

However, by our approach, each computational behavior defined by Y. Wang in [53] becomes really an algebraic object of category. In addition, imperative computing systems, adaptive computing systems, and autonomic computing systems are just instances of a functor on such the category.

- In the considered context, we apply the category theory, which deals in an abstract way with algebraic objects and relationships between them for specifying interaction behaviors in ANCSs. For modeling, analyzing, and verifying the interaction behaviors, category theory is much better-approaching than other ones such as *process algebras* (or *process calculi*), FSM (Finite State Machine), or UML (Unified Modeling Language). In fact, the categorical approach becomes more powerful since process algebras and FSM are just of algebraic objects of category and UML is really a semi-formal approach. Categories were first described by Samuel Eilenberg and Saunders Mac Lane in 1945 [29], but have since grown substantially to become a branch of modern mathematics. Category theory spreads its influence over the development of both mathematics and theoretical computer science. The categorical structures themselves are still the subject of active research, including work to increase their range of practical applicability.

10 Conclusions

In this chapter, we have rigorously approached to the notion of self-* in ANCSs from which formal aspects of the self-* emerge.

We have started with investigating self-* in DANCSs and NANCSs, where we have modeled configuration of the system at every adaptation step as a member in the set $Sys \times Context^{i \in T}$, then self-* as a morphism from a configuration to another and self-* facet as a member of self-*. Moreover, $\mathbf{Self-X}^{i \in T}$ has been constructed as a self-* monoid to shape series $T \xrightarrow{sf} \mathbf{Self-X}^{i \in T}$ of self-* facets. By the self-* monoids, we have formed $\mathbf{Cat}(\mathbf{Self-X})$ to be a category of the self-* monoids for discovering the significant properties of the self-*.

Acknowledgments Thank you to the anonymous reviewers for their helpful comments and valuable suggestions which have contributed to the final preparation of the chapter. As always, I am deeply indebted to Professor Jonathan P. Bowen, Head of the Centre for Applied Formal Methods (CAFM) at London South Bank University (LSBU) in United Kingdom, for a constant source of inspiration and encouragement for the work which culminated in the publication of this chapter.

References

1. S. Abdelwahed and N. Kandasamy. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter A Control-Based Approach to Autonomic Performance Management in Computing Systems, pages 149–168. CRC Press, 1st edition, 2006.
2. J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories*. John Wiley and Sons, 1990.
3. R. Adams et al. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Scalable Management – Technologies for Management of Large-Scale, Distributed Systems, pages 305–328. CRC Press, 1st edition, 2006.
4. R.R. Amoud et al. *Advanced Autonomic Networking and Communication*, chapter An Autonomic MPLS DiffServ-TE Domain, pages 149–168. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.

5. R. Anthony, A. Butler, and M. Ibrahim. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Exploiting Emergence in Autonomic Systems, pages 121–148. CRC Press, 1st edition, 2006.
6. A. Asperti and G. Longo. *Categories, Types and Structures*. M.I.T. Press, 1991.
7. G. M. Bergman. *An Invitation to General Algebra and Universal Constructions*. Henry Helson, 15 The Crescent, Berkeley CA 94708, USA, 1998.
8. V. Bhat, M. Parashar, and N. Kandasamy. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Autonomic Data Streaming for High-Performance Scientific Applications, pages 413–434. CRC Press, 1st edition, 2006.
9. D.W. Bustard and R. Sterritt. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter A Requirements Engineering Perspective on Autonomic Systems Development, pages 19–34. CRC Press, 1st edition, 2006.
10. W. Butera. Text Display and Graphics Control on a Paintable Computer. In G.D.M. Seruendo, J.P.M. Flatin, and M. Jelasity, editors, *Proceedings of 1st International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*, pages 45–54. IEEE Computer Society Press. Boston, MA, USA, 9–11 July 2007.
11. M. Calisti, R. Ghizzioli, and D. Greenwood. *Advanced Autonomic Networking and Communication*, chapter Autonomic Service Access Management for Next Generation Converged Networks, pages 101–126. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
12. M. Calisti, S.V.D. Meer, and J. Strassner, editors. *Advanced Autonomic Networking and Communication*. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 2008. 190 pages.
13. A. Chakravarti, G. Baumgartner, and M. Lauria. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Self-Organizing Scheduling on the Organic Grid, pages 389–412. CRC Press, 1st edition, 2006.
14. J. Chen et al. *Advanced Autonomic Networking and Communication*, chapter Game Theoretic Framework for Autonomic Spectrum Management in Heterogeneous Wireless Networks, pages 169–190. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
15. D.M. Chess, J.E. Hanson, J.O. Kephart, I. Whalley, and S.R. White. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Dynamic Collaboration in Autonomic Computing, pages 253–274. CRC Press, 1st edition, 2006.
16. L. Durham, M. Milenkovic, P. Cayton, and M. Yousif. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Platform Support for Autonomic Computing: A Research Vehicle, pages 329–350. CRC Press, 1st edition, 2006.
17. C. Fahy et al. *Advanced Autonomic Networking and Communication*, chapter Modelling Behaviour and Distribution for the Management of Next Generation Networks, pages 43–62. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
18. A. Ganek. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Overview of Autonomic Computing: Origins, Evolution, Direction, pages 3–18. CRC Press, 1st edition, 2006.
19. D. Greenwood and R. Ghizzioli. *Advanced Autonomic Networking and Communication*, chapter Autonomic Communication with RASCAL Hybrid Connectivity Management, pages 63–80. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
20. R. Griffith, G. Valetto, and G. Kaiser. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Effecting Runtime Reconfiguration in Managed Execution Environments, pages 369–388. CRC Press, 1st edition, 2006.
21. T. Heinis, C. Pautasso, and G. Alonso. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter A Self-Configuring Service Composition Engine, pages 237–252. CRC Press, 1st edition, 2006.

22. IBM. Autonomic Computing Manifesto. Retrieved from <http://www.research.ibm.com/autonomic/>, 2001.
23. B. Jacobs and J. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *Bulletin of EATCS*, 62:222–259, 1997.
24. G. Jiang et al. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Trace Analysis for Fault Detection in Application Servers, pages 471–492. CRC Press, 1st edition, 2006.
25. X. Jin and J. Liu. From Individual Based Modeling to Autonomy Oriented Computation. In M. Nickles, M. Rovatsos, and G. Weiss, editors, *Agents and Computational Autonomy: Potential, Risks, and Solutions*, volume 2969 of *Lecture Notes in Computer Science*, pages 151 – 169. Springer Berlin, April 2004.
26. B. Khargharia and S. Hariri. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Autonomic Power and Performance Management of Internet Data, pages 435–470. CRC Press, 1st edition, 2006.
27. W. Kinsner. Towards Cognitive Machines: Multiscale Measures and Analysis. *The International Journal on Cognitive Informatics and Natural Intelligence (IJCINI)*, 1(1):28–38, 2007.
28. S. Ko, I. Gupta, and Y. Jo. Novel Mathematics-Inspired Algorithms for Self-Adaptive Peer-to-Peer Computing. In G.D.M. Serugendo, J.P.M. Flatin, and M. Jelasity, editors, *Proceedings of 1st International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*, pages 3–12. IEEE Computer Society Press. Boston, Massachusetts, USA, 9–11 July 2007.
29. F.W. Lawvere and S.H. Schanuel. *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, 1st edition, 1997.
30. M. Levine. Categorical Algebra. In G. Benkart, T.S. Ratiu, H.A. Masur, and M. Renardy, editors, *Mixed Motives*, volume 57 of *Mathematical Surveys and Monographs*, chapter I, II, II of Part II, pages 373–499. American Mathematical Society, USA, 1998.
31. H. Liu and M. Parashar. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter A Programming System for Autonomic Self-Managing Applications, pages 211–236. CRC Press, 1st edition, 2006.
32. J.A.L. López, J.M.G. Munoz, and J.M. Padial. *Advanced Autonomic Networking and Communication*, chapter A Telco Approach to Autonomic Infrastructure Management, pages 27–42. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
33. S.V.D. Meer et al. *Advanced Autonomic Networking and Communication*, chapter Technology Neutral Principles and Concepts for Autonomic Networking, pages 1–25. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
34. D.A. Menascé and M.N. Bennani. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Dynamic Server Allocation for Autonomic Service Centers in the Presence of Failures, pages 353–368. CRC Press, 1st edition, 2006.
35. G. Nguengang et al. *Advanced Autonomic Networking and Communication*, chapter Autonomic Resource Regulation in IP Military Networks: A Situatedness Based Knowledge Plane, pages 81–100. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
36. O. Pacheco. Autonomy in an Organizational Context. In M. Nickles, M. Rovatsos, and G. Weiss, editors, *Agents and Computational Autonomy: Potential, Risks, and Solutions*, volume 2969 of *Lecture Notes in Computer Science*, pages 195–208. Springer Berlin, April 2004.
37. M. Parashar. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Autonomic Grid Computing: Concepts, Requirements, and Infrastructure, pages 49–70. CRC Press, 1st edition, 2006.
38. M. Parashar and S. Hariri, editors. *Autonomic Computing: Concepts, Infrastructure and Applications*, 568 pages. CRC Press, 1st edition, December 2006.
39. G. Qu and S. Hariri. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Anomaly-Based Self Protection against Network Attacks, pages 493–522. CRC Press, 1st edition, 2006.

40. M.A. Razzaque, S. Dobson, and P. Nixon. *Advanced Autonomic Networking and Communication*, chapter Cross-layer Optimisations for Autonomic Networks, pages 127–148. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer-Verlag, 1st edition, 2008.
41. R.V. Renesse and K.P. Birman. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Autonomic Computing: A System-Wide Perspective, pages 35–48. CRC Press, 1st edition, 2006.
42. J.J.M.M. Rutten. Universal Coalgebra: A Theory of Systems. *Theoretical Computer Science*, 249(1):3–80, 17 October 2000.
43. J.J.M.M. Rutten. Elements of Stream Calculus (An Extensive Exercise in Coinduction). *Electronic Notes in Theoretical Computer Science*, 45, 2001. Elsevier Science Publishers Ltd.
44. S.M. Sadjadi and P.K. McKinley. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Transparent Autonomization in Composite Systems, pages 169–188. CRC Press, 1st edition, 2006.
45. K. Schwan et al. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter AutoFlow: Autonomic Information Flows for Critical Information Systems, pages 275–304. CRC Press, 1st edition, 2006.
46. P. Steenkiste and A.C. Huang. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Recipe-Based Service Configuration and Adaptation, pages 189–208. CRC Press, 1st edition, 2006.
47. J.W. Sweitzer and C. Draper. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter Architecture Overview for Autonomic Computing, pages 71–98. CRC Press, 1st edition, 2006.
48. P.C. Vinh. *Formal Aspects of Dynamic Reconfigurability in Reconfigurable Computing Systems*. PhD thesis, London South Bank University, 103 Borough Road, London SE1 0AA, UK, 4 May 2006.
49. P.C. Vinh. Homomorphism between AOMRC and Hoare Model of Deterministic Reconfiguration Processes in Reconfigurable Computing Systems. *Scientific Annals of Computer Science*, (XVII):113–145, 2007.
50. P.C. Vinh and J.P. Bowen. A Formal Approach to Aspect-Oriented Modular Reconfigurable Computing. In *Proceedings of 1st IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 369–378. IEEE Computer Society Press. Shanghai, China, 6–8 June 2007.
51. P.C. Vinh and J.P. Bowen. Formalization of Data Flow Computing and a Coinductive Approach to Verifying Flowware Synthesis. *LNCS Transactions on Computational Science*, 1(4750):1–36, June 2008.
52. Y. Wang. Exploring machine cognition mechanisms for autonomic computing. *The International Journal on Cognitive Informatics and Natural Intelligence (IJCINI)*, 1(2):i–v, 2007.
53. Y. Wang. Toward Theoretical Foundations of Autonomic Computing. *The International Journal of Cognitive Informatics and Natural Intelligence (IJCiNi)*, 1(3):1–16, July–September 2007.
54. M. Witkowski and K. Stathis. A Dialectic Architecture for Computational Autonomy. In M. Nickles, M. Rovatsos, and G. Weiss, editors, *Agents and Computational Autonomy: Potential, Risks, and Solutions*, volume 2969 of *Lecture Notes in Computer Science*, pages 261–273. Springer Berlin, April 2004.
55. T.D. Wolf and T. Holvoet. *Autonomic Computing: Concepts, Infrastructure and Applications*, chapter A Taxonomy for Self-* Properties in Decentralized Autonomic Computing, pages 101–120. CRC Press, 1st edition, 2006.
56. B. Yang and J. Liu. An Autonomy Oriented Computing (AOC) Approach to Distributed Network Community Mining. In G.D.M. Serugendo, J.P.M. Flatin, and M. Jelasity, editors, *Proceedings of 1st International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*, pages 151–160. IEEE Computer Society Press, Boston, MA, USA, 9–11 July 2007.

Autonomic Information Diffusion in Intermittently Connected Networks

Sara Alouf, Iacopo Carreras, Álvaro Fialho, Daniele Miorandi,
and Giovanni Neglia

Abstract In this work, we introduce a framework for designing autonomic information diffusion mechanisms in intermittently connected wireless networks. Our approach is based on the use of techniques and tools drawn from evolutionary computing research, which enable to embed evolutionary features in epidemic-style forwarding mechanisms. In this way, it is possible to build a system in which information dissemination strategies change at runtime to adapt to the current network conditions in a distributed autonomic fashion. A case study is then introduced, for which design and implementation choices are presented and discussed. Simulation results are reported to validate the ability of the proposed protocol to converge to the optimal operating point (or close to it) in unknown and changing environments.

1 Introduction

Epidemic-style forwarding [13] has been proposed as an approach for achieving systemwide dissemination of messages in intermittently connected networks [10] (sometimes named also Delay-Tolerant Networks (DTNs) [3]). These networks are sparse and/or highly mobile wireless ad hoc networks where no continuous connectivity guarantees can be assumed. Epidemic-style forwarding is based on a “store-carry-forward” paradigm: a node receiving a message buffers and carries that message as it moves, passing it on to new nodes upon encounter. Alike the spread of infectious diseases, each time a message-carrying node encounters a new node not having a copy thereof, the carrier may decide to *infect* this new node by passing on a message copy; newly infected nodes, in turn, behave similarly. The destination receives the message when it first meets an infected node.

S. Alouf (✉)
INRIA, Sophia Antipolis, France
e-mail: sara.alouf@sophia.inria.fr

An unconstrained epidemic forwarding scheme (in which an infected node spreads the messages to all nodes it encounters) is able to achieve minimum delivery delay at the expense of an increased use of resources such as buffer space, bandwidth, and transmission power. Variations of epidemic forwarding have been recently proposed in order to exploit the trade-off between delivery delay and resource consumption. This family includes probabilistic forwarding [7], K -hop schemes [4], and spray routing [12].

Depending on the specific application scenario, different performance metrics could be envisaged, such as the probability of successfully delivering a message to the destination, the delivery time, the total energy consumption in the system or a combination of the previous ones. For a given optimization goal, the choice of a specific forwarding scheme and its parameters configuration depend in general on the number of nodes in the system, on their mobility patterns and on the traffic generated in the networks [8]. In many scenarios, these characteristics cannot be known at system design and deployment time and may drastically change across time and space.

In order to deal with these issues, various adaptive techniques for message forwarding can be envisaged. Conventional approaches are limited in that they require an a priori definition of the actions to be taken to optimize the mechanism for some specific situation. In this chapter, we propose a novel approach, based on the inclusion of autonomic features (in the form of self-optimization capabilities) in the forwarding service itself. This is achieved by using concepts and tools borrowed from the Evolutionary Computation (EC) field. Self-optimization is obtained through a cooperative process, in which nodes exchange their knowledge on the performance of the various schemes employed.

A fundamental observation is that forwarding schemes simply perform a decision whether to relay a copy of a given message to an encountered node or not. Thus, multiple forwarding schemes can coexist and interact within the same network. Each node can then employ a potentially different forwarding policy, which prescribes the operations to be undertaken when receiving a message destined to another node. We assume that a way to formally describe forwarding policies is available (it could be as simple as a list of parameters). We call such description the *genotype* of the forwarding policy. Genotypes are associated with a fitness measure which, roughly speaking, indicates the ability of the current set of parameters to achieve good performance in the local environment with respect to the predefined optimization goal. Fitness estimation is probably the most challenging task in this autonomic service. In fact a node cannot evaluate by itself whether its current policy fits the current scenario, because (1) it is in general not aware of the consequences of its actions (for example a given node may be relaying a message which has already been delivered to its destination) and (2) the fitness of a node's policy depends on the policies implemented by the other nodes as well. Fitness is estimated at each node using not only local information but also feedback from the destination. The fitness estimation process takes into account both feedback delay and feedback noise (due to randomness of the mobility and arrival processes). When two nodes meet, they may exchange genotypes (and associated fitness levels), updating the pool they

maintain. Each node periodically generates a new genotype, judiciously using those in its pool. By changing its genotype, a node has made its policy *evolve*. The use of genotypes from other nodes in order to generate a new genotype implements a form of distributed learning, being that each node takes advantage of other nodes' experience. Standard EC operators such as selection, crossover, and mutation are used to generate a new genotype from the pool, but also other approaches are considered. The whole system is engineered in such a way to present a drift toward higher fitness levels. The performance of the proposed scheme is evaluated through extensive simulations, showing its ability to adapt, in a seamless and autonomic way, to varying system conditions.

The remainder of the chapter is organized as follows. Section 2 reviews the related works and motivates our approach. Section 3 describes the autonomic information diffusion service. Section 4 illustrates a case study implementation of the proposed approach to autonomic epidemic-style forwarding in intermittently connected networks. A detailed simulative study, performed using a freely available software tool, provides important insights in the applicability of the proposed approach. Section 5 concludes the chapter.

2 Related Works and Motivation

Intermittently connected wireless systems represent a challenging environment for networking research, due to the problems of ensuring messages delivery in spite of frequent disconnections and random meeting patterns. Due to the mobility of the nodes, protocols such as Ad hoc On-demand Distance Vector (AODV) routing, Dynamic Source Routing (DSR), or Optimized Link State Routing (OLSR), continuously update routes when users require them (AODV and DSR) or in a proactive way (OLSR). These routes commonly time out after a few seconds. When a path between two nodes does not exist through the network, no route can be created. Needless to say that these protocols can hardly run over DTNs and will fail to deliver data most of the time, because the assumption on the existence at a given time of a complete path between a source and a destination is simply not met.

Many solutions have been proposed for use in such environments over the past few years. The common basis of these solutions is the observation that end-to-end routes may exist over time due to nodes mobility: leveraging their mobility, nodes can exchange and carry other node messages upon meetings, and deliver them afterward to their destinations. This novel routing paradigm is referred to as *store-carry-forward*. Each node in a network serves then as a relay for all other nodes. Different approaches have been proposed depending if contacts among nodes can be planned, predicted, or are unknown in advance. We want to provide a solution for the third case, without relying on any infrastructure or special mobile nodes. We will then briefly review other solutions present in literature for such a scenario.

All of these solutions pertain to the family of epidemic-like forwarding, as they all imitate the spread of viruses in nature. A node having a copy of the message

is said to be *infected*, and as such, it can infect any other non-infected nodes that it encounters by passing a message copy to it. If the message is copied at every meeting, the delivery delay is the shortest possible. However, this scheme is extremely wasteful of resources like the channel capacity, the buffer space and the energy. We refer to this protocol as the unconstrained epidemic forwarding scheme.

Vahdat and Becker [13] are the first to propose such a scheme, and call it “epidemic routing.” Messages are simply flooded in the network, the only limitation being the maximal number of hops done by a message. The authors show that (i) epidemic routing delivers all packets given unbounded buffer size at each node and (ii) by appropriately choosing the maximum hop count, delivery rates can still be kept high while reducing resource utilization.

To improve over the simple flooding achieved by epidemic routing, Lindgren, Doria, and Schelén propose PROPHET [7], a probabilistic forwarding scheme that is more sophisticated than [13]. Using history of node encounters and transitivity, PROPHET achieves a performance comparable to that of epidemic routing but with a lower communication overhead.

Groenevelt, Nain and Koole introduce in [4] the multi-copy two-hop relay protocol, a variant of the single-copy two-hop relay protocol proposed and studied in [5]. The infection process is largely constrained as any node can be infected only by the source and can itself infect only the destination.

A new family of routing protocols is proposed in [12] by Spyropoulos, Psounis, and Raghavendra. This family, called *Spray routing*, can be viewed as a trade-off between single and multiple copies techniques. Spray routing consists of two phases: the first is called *spray*, and the second is either *wait* (spray-and-wait protocol) or *focus* (spray-and-focus protocol). In the spray phase, a carefully chosen number of copies of the message are generated and disseminated in the network to the same number of relay nodes. In the wait phase, relays simply wait to meet the destination in order to deliver the message. In the focus phase, each copy of the message is routed according to a utility-based single-copy routing algorithm. The authors show that, if carefully designed, spray routing incurs significantly fewer transmissions per message than epidemic routing, and achieves a trade-off between efficient message delivery and low overhead.

Epidemic schemes may be combined with a so-called recovery process that deletes copies of a message at infected nodes, following the successful delivery of the message to the destination. Different recovery schemes have been proposed: some are simply based on timers, others actively spread in the network the information that a copy has been delivered to the destination [6].

In this work, we focus on data dissemination techniques which do not require nodes to exchange any a priori knowledge/estimation of their meeting patterns and/or location. We are interested in epidemic-style forwarding policies due to their simplicity, inherent robustness with respect to node failures and unpredictable system conditions, as well as totally distributed nature. The major drawback of these schemes is that they can potentially harvest the network resources. Also, epidemic-style forwarding is highly sensitive to the setting of some protocol parameters

(e.g., the forwarding probability), which in turn should be set according to the current operating conditions. This motivates us to develop an evolutionary forwarding service that will adapt to changes in the network conditions.

3 The Framework

In this work we aim to develop a framework that will allow the forwarding service to evolve online in order to optimize a given performance metric and to adapt autonomously to the actual system operating conditions.

We first observe that multiple forwarding schemes can coexist at the same time in the network. In fact this form of information delivery, based on the presence of multiple copies in the network, does not need nodes to be compliant with a specific behavior, enhancing system robustness with respect to conventional schemes. This flexibility comes from the completely distributed nature of the forwarding process in epidemic-style relaying, which allows nodes to use different policies in an uncoordinated fashion.

In our framework, we want nodes to “learn” online what is the best forwarding policy (or what are good policies) in the current scenario and change consequently the one they employ. The problem is challenging as a node cannot evaluate by itself whether its current policy fits the current scenario, because it is in general not aware of the consequences of its actions. For example a given node can never know by itself whether its decisions—according to its forwarding policy—to relay or not to relay a message were the right ones or not. Thus, a node may be relaying a message when the latter has already been delivered to its destination, hence wasting resources. On the other hand, a node may refrain from relaying a message when it happens to be the key node in the message delivery process, e.g., if it is the only node traveling between two disconnected clusters of nodes in the network. It should be clear therefore that a cooperative fitness estimation process has to be put in place in order to allow the network to show self-optimizing features. We also observe that the goodness (or fitness) of a node’s policy depends on the policies implemented by the other nodes as well. Message delivery is in fact a collaborative process, whose performance depends on the behaviors of all nodes, so that a specific policy can be beneficial or detrimental depending on other nodes actions.

The previous considerations imply the need of an online distributed fitness evaluation process and raise an issue about the use of the fitness in the evolution process. Once a node get marks for its own policy, how should these marks be used in order to change the policy? We opt for a blind evolutionary approach, which relies on a *homogeneity* assumption, according to which the nodes in the network can be partitioned in “large” groups of homogeneous ones, having similar mobility models and traffic patterns. If this assumption holds, then each node can learn from nodes in the same group: it can make its policy more similar to those policies presenting a higher level of fitness. This suggests that two other components are required in our framework: a unified description of the policies, so that each node can communicate

to other nodes the one in use, and mechanisms for the generation of new (hopefully better) policies from existing ones.

Here we summarize the three fundamental components in our evolutionary framework using EC terminology. These components are as follows:

1. the possibility to share with other nodes a description of the specific forwarding mechanism deployed at each node (the *genotype* associated to the forwarding policy employed at the node);
2. a consistent process for evaluating the *fitness* of the schemes employed, in order to identify “good” solutions;
3. the possibility for each node to modify its forwarding scheme taking into account the schemes of other nodes (what we call, with a slight abuse of terminology, the genotype *evolution*).

In order to enhance the readability, the notation used in this chapter is summarized in Table 1.

Table 1 Notation

x_i	genotype of node i
F	function to optimize
f	performance metric of interest
CF	cost function
P_i	forwarding probability in genotype of node i
H_i	maximum allowed hop-count in genotype of node i
$T_D^{(n)}$	delivery time of message n
$W^{(n)}$	set of nodes in the path of the first copy of message n reaching destination
h_n	hop count of the first copy of message n reaching destination
$C_i^{(n)}$	number of copies of message n done by node i
$C^{(n)}$	total number of copies of a message n in the system
$\hat{C}_i^{(n)}$	estimation at node i of the total number of copies of message n in the system
γ	time-equivalent cost of a copy in the cost function
$R_i^{(n)}$	reward received by node i for spreading message n
ϕ_i	fitness of node i genotype
$\hat{\phi}_i$	estimation of node i genotype fitness
$\hat{\phi}_{i,j}$	estimation of node i genotype fitness known by node j
G_i	set of genotypes in the pool of node i
$p_{i,j}$	probability of selecting genotype x_i at node j during the reproduction phase
T_g	time between two consecutive reproductions
p_c	crossing-over probability
p_m	mutation probability
N	number of mobile nodes
T_s	maximum inter-message arrival time at each node
L	side size of squared playground
r	transmission range
v	node speed
T_{step}	mobility time step in simulations

3.1 The Forwarding Policy

A first step toward an evolutionary delay-tolerant forwarding service consists in a formal representation of a generic forwarding scheme. Such descriptions represent the *genotypes* of the implemented forwarding schemes.

In order to define this formal description, it is essential to identify the key actions potentially involved in a message diffusion process. Whenever two nodes become within each other's transmission range, they have the possibility to exchange some or all of their messages. Regarding the transmission of messages, each of the nodes can perform one of the following actions upon each of its messages: (i) transmitting a copy of the message; (ii) transferring the message itself without keeping a local copy; or (iii) retaining the message. These actions can be further specified and limited by counters and timers or can be performed according to some probability distribution. The messages can be processed on a first come first served basis or according to any other scheduling policy. It is also possible that nodes base their actions on some information contained in message headers (like message generation time or number of hops the message has traversed) and also update it. Regarding the reception of messages, a node receiving a message may face an overflow in its buffer. When this occurs, the node needs to drop some messages from its buffer in order to free space for the new message. Messages to be dropped can be selected at random or according to a specific criterion (the oldest, the most spread, etc.).

All these possible schemes could be specified in different ways, through array of parameters, variable length strings, trees, etc. In what follows we briefly describe two possibilities.

3.1.1 Parameters Arrays

We can consider a complex forwarding scheme, which can implement different known schemes, by tuning its parameters. For example, the scheme could select the message to forward from the queue according to a probability distribution and copy it only if a local copies counter is below a given threshold and the messages has not traversed more nodes than the value specified by another threshold. This scheme is univocally characterized by its array of parameters including the probability distribution, the copies threshold and the hops threshold. Clearly the scheme could be made more complex and a longer array would be needed.

While this kind of description can become heavy, working with fixed length genotypes simplifies genotype interpretation and genotype processing at each node.

3.1.2 Strings

Another possibility is to rely on strings. The purpose of this section is not to define a formal language but to introduce some considerations and provide some examples.

The genotype string can be divided into two parts: a first part meaningful in transmission mode and specifying the actions to undertake and their parameters when transmitting a message; and a second part meaningful in reception mode and specifying the actions to undertake and their parameters when the buffer is full.

According to the description of nodes operation, we can define many basic genes, which specify actions and their characteristics or parameters. For illustrative purposes we define the following:

- action* genes:
 - Select the message (S),
 - Transmit a copy of the message (T),
 - transFer the message (F),
 - Drop the message (D);
- parameter* genes:
 - relaying Nodes counter (N),
 - Copies counter (C),
 - with Probability (P);
- and *location* genes:
 - uniformly Random (R),
 - at the Head (H),
 - at the Bottom (B).

Genes (N), (C), and (P) limit the actions of genes (T) and (F) whereas genes (R), (H), and (B) accompany gene (S). If we assume that messages in the queue are chronologically ordered having the most recent one at the bottom of the queue, then genes (H) and (B), respectively, represent FIFO and LIFO service disciplines. Note that we could have added a third part to the string representation to define how messages are ordered inside the buffer. However, we refrained from doing this so as to lighten the string representation.

As we said we are not going to specify the strings grammar, but we present some possible examples of strings. The purpose is to show how these few genes can yet be sufficient to specify many different behaviors. The two parts of the string will be separated by a dash and genes representing a value will be put between parentheses. Examples are

SBT-SHD: This string describes the standard epidemic routing with priority for recent messages (the node forwards most recent messages and discards least recent ones);

SBTP(0.1)-SRD: This string describes probabilistic forwarding of most recent messages with a forwarding probability equal to 0.1, and where discarded messages are picked uniformly at random;

SRTN(10)-: This string describes a K -hop scheme which limits the diffusion of messages up to the tenth relaying node and where forwarded messages are picked uniformly at random. The absence of the second part could correspond to the case where incoming messages are discarded when the buffer is full (no message already in the buffer would be dropped).

While it does not seem difficult to create a formal language to include such simple examples, the trade-off between the complexity of the description and the variety

of possible schemes which can be described (the possible *phenotypes*) has to be investigated.

3.2 The Selection Process

The natural selection process promotes the diffusion of organisms presenting a high fitness level. In much the same way, we want to engineer mechanisms for promoting the diffusion of the genotypes yielding good performance for the optimization goal. At the same time, new genotypes can be generated in order to explore new possible solutions. In this section, we focus on the first issue letting the next section discuss how new genotypes can be generated from existing ones.

In traditional EC, and in particular in Genetic Algorithms (GAs), *reproduction* is a process which selects existing genotypes to create new offsprings. At discrete time intervals, genotypes are randomly selected from the population to generate offsprings: genotype x_i is selected with probability proportional to its fitness, namely, $p_i := \phi_i / \sum_j \phi_j$. This procedure tends by itself to increase the average fitness of the population. However, in the considered mobile network scenario the different genotypes are distributed over all the nodes of the network. This means that standard GAs reproduction phase can not occur without resorting to a centralized solution where a central node (i) stores the genotypes, (ii) applies GA operators, and (iii) distributes back the produced offsprings to the mobile nodes. In order to overcome this limitation and devise a *distributed* solution, we assume that upon meeting nodes exchange information about their respective genotypes and the corresponding fitness indexes. As depicted in Fig. 1, in our system each node maintains a pool of available genotypes (including the one currently in use) and their corresponding fitness values. At regular time intervals, each node goes into a reproduction phase, running the selection process on the genotypes currently stored in its own pool.

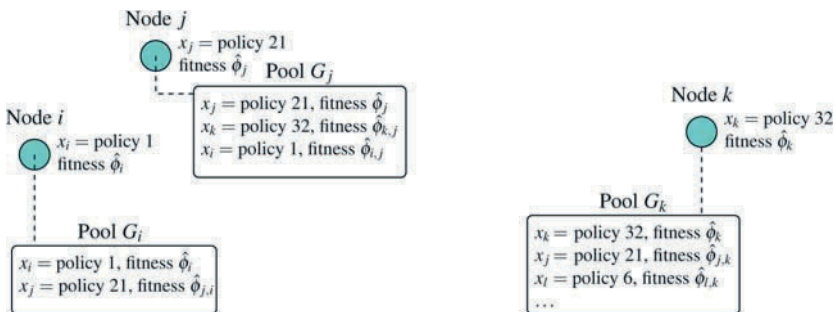


Fig. 1 Considered system architecture: each node employs a policy, characterized by a genotype with associated fitness value. Each node maintains a pool of candidate solutions used in the reproduction phase to generate new genotypes

3.3 Fitness Evaluation

We now need to define formally the fitness of a forwarding policy. For the sake of simplicity, we consider that the function to optimize, F , is the expected value of some performance metric, say f , which can be evaluated for a specific infection process. More formally let I refer to the complete history of a generic infection process in the network. An infection I_n reports all of the infection steps since the generation of message n until the cancellation of the message and all of its copies. For example it specifies which nodes are infected at a given time instant. The infection I_n depends on the genotypes and the mobility patterns of all nodes involved in the infection process but also on the concurrent data traffic at these nodes. The function that we want to optimize can be rewritten as $F = \mathbb{E}[f(I)]$ where the expectation is taken with respect to the probability measure defined by the mobility process, the message generation process. Examples of performance metric $f(I)$ are the time needed to deliver a message to the intended destination, the time before an infection dies (i.e., when all copies are erased from the network), the number of copies done for a given message and the power required to propagate the message.

Observe that a given infection I will most likely involve only a subset of the nodes in the system. Conversely, a given node j will take part in only a subset of all infections. The fitness of a genotype x_j can be defined then as

$$\phi_j = \mathbb{E}[f(I) \mid \text{node } j \text{ contributed to infection } I]. \quad (1)$$

This ensures that the genotypes of nodes taking part in “good” delivery processes get on average a higher fitness than those involved in “bad” diffusion processes.

According to Eq. (1), for a node to estimate the fitness of the genotype it is using, it should average the performance metric $f(I)$ over all infections it had taken part in. A difficulty arises from the fact that the forwarding service is intrinsically a cooperative distributed service and a node is in general not able to evaluate by itself $f(I)$. For example, the node does not know when or how many times a message is copied in the system, whether or not a message has been delivered, and so on. Some signaling among nodes is thus required in order to let each node be able to evaluate $f(I)$.

In many cases, the process of evaluating $f(I)$ can be triggered by the destination node of a message as it is the best entitled to evaluate the outcome of the infection process. The destination node propagates then the evaluation of $f(I)$ or at least information needed for this evaluation to all nodes involved in the infection process. This feedback can be seen as a “reward” to those nodes. The communication cost can become significant so that a communication-cost versus information-accuracy trade-off arises. Beside the communication overhead, another aspect to consider is the time needed to evaluate the performance metric $f(I)$. If information is delivered to a node long after message delivery, the node could have changed its genotype, so that the evaluation would not refer to the current genotype.

3.4 Generation of New Forwarding Policies

New forwarding policies are generated applying EC-like operators to the genotypes maintained by a node in its pool. The two following operators can be used to create new genotypes from existing ones. *Crossing-over* consists in breaking two genotypes at a randomly chosen position and exchanging the tails of the genotypes. Two offsprings, called *crossovers*, are produced, and one is selected at random. *Mutation* consists in a random change occurring in the genotypes. As an example, mutation can be implemented by randomly swapping, with some probability, the bits of a binary representation of the genotype, or by adding some white noise to the protocol genotype. To ensure stability, mutation should occur with small probability.

4 The Case Study

In this section, we report on a case study implementation of the proposed approach to epidemic-style forwarding in DTNs. Our main objective with this implementation is to gain insight into the applicability of the approach proposed in Sect. 3. In particular, our purpose is to answer the following questions:

- (i) Does the distributed genetic algorithm “converge”?
- (ii) If so, what does the convergence point look like and how much time is required for the convergence?
- (iii) What is the impact of the mutation process on the speed of convergence?
- (iv) How robust is our scheme and how capable is it of adapting to changing network conditions?

These questions will be tackled by implementing a (reduced) version of the proposed framework, and running numerical simulations to evaluate, in a realistic scenario, the behavior of the system.

4.1 Description

We consider a simple fixed-length genotype comprising one parameter, which is the probability P to copy a message upon encountering a new node. In our prior work [1], a simple binary string was used to represent the value of P . In this work, P is maintained as a double-precision number. As optimization goal we consider the minimization of the expectation of the weighted sum of the delivery time, T_D , and the number of copies of the message done in the system, C . The cost function is then defined as follows:

$$CF = \mathbb{E} [T_D + \gamma C], \quad (2)$$

where γ is a parameter which can be understood as the time-equivalent cost of a copy.

Should the optimization goal be to minimize solely the expected delivery time, then the evolutionary forwarding scheme will trivially converge, in an underloaded network—i.e., when traffic is small in comparison to the available capacity—to standard epidemic routing, where messages are flooded in the entire network. Conversely, the presence of the number of copies in the cost function makes also an underloaded network (a realistic case and faster to simulate) an interesting scenario to study. Such a metric is also meaningful as it is strongly related to bandwidth usage and power consumption. The evolutionary forwarding scheme will limit the number of copies to an extent that depends on the value of the parameter γ .

4.1.1 Fitness

According to the discussion in Sect. 3.2, we can think to define the fitness of the genotype at node j as

$$\phi_j = \mathbb{E} \left[\left(1 - \frac{CF}{R_{\max}} \right) \mid \text{node } j \text{ contributed to infection} \right].$$

In this way the fitness is a decreasing function of the cost and if we let R_{\max} be a high enough value, we can guarantee that fitness values are positive, as it is required for the biased selection process we described above. We have decided to consider as nodes contributors to the infection of a given message, say message n , only those in the forwarding path of the first copy that reaches the destination. This set of nodes is denoted as $W^{(n)}$. This choice reduces the communication burden in comparison to considering all infected nodes.

Let us discuss now how a node can estimate its fitness. This estimation clearly requires some information about the delivery time and the number of copies done for each message n for which the node is in the set $W^{(n)}$. Regarding the delivery time, we assume that all nodes are synchronized and that the message header contains a field specifying the time at which the payload was generated at the source.¹ In such a way the destination can evaluate the delivery time as soon as it receives the first copy of a message. On the other hand, each node knows the number of times it has copied a message, but not the delivery time. We assume that each node, before forwarding a copy of a message, say message n , adds its own identifier (ID) to the message header (this is analogous to what is done in source routing in mobile ad hoc networks). It follows then that the IDs in the header of the first copy of message n reaching the destination identify exactly the nodes in the set $W^{(n)}$. The destination node sends to these nodes a new acknowledgment (ACK) message. This message

¹ If local clocks are enough accurate at the message delivery timescale, then there is another solution which does not require synchronization. Message header should have a field which indicates the time since the payload was generated. This field can be updated by each node before forwarding the message. In this case the node should keep track of the time running since it has received the message.

specifies the delivery delay and the number of hops ($h_n = |W^{(n)}|$) traveled by the message before reaching the intended destination.

Estimating the total number of copies is a more complex issue. In the simulation results shown later we assume that nodes know the number of copies done in the system when they receive the ACK and use this number as an estimate of the total number of copies, which will in general be larger being that the infection can still be propagating. We discuss briefly realistic estimation approaches, that we are evaluating. Each node can keep track of the number of copies it did for message n , let us denote as $C_j^{(n)}$ the number of copies done by node j . An approach is then to let each node broadcast its local number of copies and then evaluate the total number of copies aggregating the information received by other nodes ($C^{(n)} = \sum_{j=1}^N C_j^{(n)}$), but this would imply a significant communication overhead. We think it is better to rely on an estimation by adopting the ‘‘Plain Diffusion’’ solution proposed by [2]. In this algorithm, two meeting nodes exchange their current estimates and evaluate a new (better) estimate. The estimations at all nodes converge with probability one to the real value, but this requires some time after the end of the infection, hence nodes need to wait after the reception of the ACK to rely on an accurate estimate.

Once a node, say node j , knows the delivery time and has a reliable estimation (say $\hat{C}_j^{(n)}$) of the total number of copies for message n , it can estimate the cost of spreading the message as $T_D^{(n)} + \gamma \hat{C}_j^{(n)}$, a quantity that we refer to as the reward $R_j^{(n)}$, and then update its fitness. Let us denote as M_j the set of all infections node j has been contributing to, formally $M_j = \{n | j \in W^{(n)}\}$. A naive approach to estimate the fitness would be to simply average the rewards over all messages in M_j . Hence we could think to estimate the fitness of node j as $1 - 1/|M_j| \sum_{n \in M_j} R_j^{(n)} / R_{\max}$. In reality, this approach would introduce a bias, because infections with longer forwarding paths, would generate a higher number of rewards in the system. The way to eliminate this bias is to consider the following weighted average:

$$\hat{\phi}_j = 1 - \frac{1}{R_{\max}} \frac{\sum_{n \in M_j} \frac{R_j^{(n)}}{h_n}}{\sum_{n \in M_j} \frac{1}{h_n}}.$$

4.1.2 Evolution

When two nodes meet, they transmit to each other their own genotype and its current fitness level estimation. Each node maintains a pool of available genotypes (including the one currently in use) and their fitness. We use G_j to denote the pool at node j and $\hat{\phi}_{i,j}$ to denote the value of node i fitness known by node j ($\hat{\phi}_{i,j}$ is the value of $\hat{\phi}_i$ at the time of the last meeting between node i and node j , so at a given time instant these two values can be different). We consider a synchronized reproduction phase. Every T_g seconds, the *generation lifetime*, nodes synchronously create a new offspring each, i.e., they update their own genotype. This synchronism

allows to clearly identify different generations during the evolution. No crossover is performed, but only mutation, in the form of addition of a properly defined white noise (described below).

At each node, e.g., at node j , the genotype to be used as a basis for the new generation is selected with a probability proportional to its own fitness,² namely, $p_{i,j} = \hat{\phi}_{i,j} / (\sum_l \hat{\phi}_{l,j})$ where the sum is over all genotypes contained in the pool G_j . Finally, one genotype selected from the pool can be mutated with probability p_m , before becoming the active genotype/forwarding policy of the node during the next generation. The genotype pool is emptied after every generation. If the network is large, the node's pool may not be large enough to keep all genotypes discovered in a generation. Should this be the case, a node may keep only the fittest genotypes, or alternatively select the genotypes according to a fitness-biased distribution.

As mentioned in Sect. 3, for stability reasons, a mutation should occur only with some small probability p_m . In [1], we have used a binary representation of the genotype and have implemented mutation by randomly swapping, with some probability, the bits of the binary representation. In this case study, we are considering a *continuous* representation of the genotype (the forwarding probability P). A mutation is performed by summing to the value P a randomly generated "noise." We want the noise to satisfy the following requirements:

1. finer grain for smaller values; this implies (i) a finer grain for negative noise than for positive noise and (ii) a smaller noise for smaller values;
2. zero mean noise;
3. negative and positive noise that are equally probable.

Requirements 2 and 3 are consistent with the well-known additive Gaussian noise, however this type of noise does not satisfy the first requirement. The latter is needed because of the increased sensitiveness of the cost function to lower values of P .

It turns out not to be possible to satisfy all three requirements simultaneously. Therefore we have decided for a distribution that satisfies the first two ones.

One simple way of achieving the first requirement is to consider the additive noise to (i) be *proportional* to the value P and (ii) have a distribution with smaller support at negative values. Let Y be a random variable having its probability density function as depicted in Fig. 2. Given that P is the value of the genotype that is selected from the pool, the value of the new genotype is defined as $(1 + Y)P$. Clearly, if $\mathbb{E}[Y] = 0$ then $\mathbb{E}[(1 + Y)P] = \mathbb{E}[P]$. The density of Y has been

² In practice, scaled fitness values are used instead of the raw fitness values $\hat{\phi}_{i,j}$. The purpose is twofold. First, we want to avoid having in the first generations few extraordinary genotypes taking over a significant proportion of the finite population in a single generation. Second, should the best fitness values be close to the average ones, we would like to have substantially more best genotypes than average ones in future generations. We can linearly scale the fitness values such that the best fitness value is double the average fitness value which remains unchanged. In generations where the scaling produces negative normalized fitness values, an alternative scaling is used. The latter maintains equality of the raw and scaled average fitness values, but maps the minimum raw fitness to a null value.

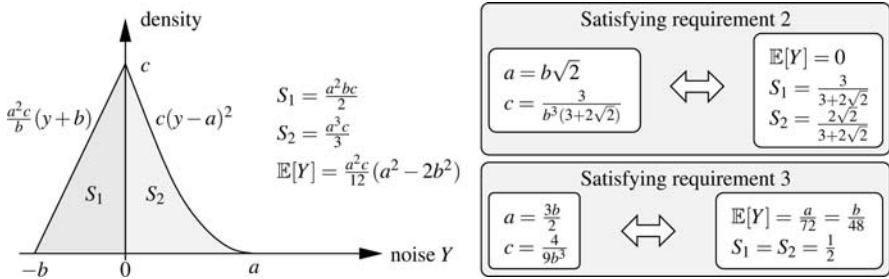


Fig. 2 Probability density function of the noise and values of the parameters a, b , and c for the satisfaction of requirements 1 and either 2 or 3

defined over the interval $[-b, a]$, where $0 < b < a < 1$. Its particular shape ensures that it is continuous at 0 and that the distribution is as balanced as possible (i.e., $S_1 := P(Y < 0)$ and $S_2 := P(Y > 0)$) as close as possible). To satisfy the second requirement, we let $a = b\sqrt{2}$; see details in Fig. 2, which also shows which values would satisfy the third requirement. In this case, $S_1/S_2 \approx 1.06066$, thus the third requirement is only slightly violated. To satisfy the third requirement, one would let $a = 3b/2$. However, the expected noise will be positive as expressed in Fig. 2. There will then be a bias toward higher values.

From now on, we consider $a = b\sqrt{2}$ and $c = 3/(b^3(3 + 2\sqrt{2}))$.

One problem is related to the setting of boundaries, as the forwarding probability P should be in the range $[0, 1]$. In our case, we have to deal only with the boundary 1, as the genotype can get only infinitely close to 0 but never below 0. Three methods to deal with boundaries are seen in the literature:

1. iterate until the value falls within the boundaries;
2. try once and bounce the value between the boundaries until it falls between them;
3. try once and truncate the value to the boundary.

All three methods introduce a bias toward smaller values in the distribution of P . The third method is expected to have the smallest bias. In particular, a mutation with a *positive* noise moving the probability beyond the boundary 1 results, with the third method, in a new value (equal to 1) that is *larger* than the original probability. Should any of the first two methods be used, there is no guarantee that a positive noise raising a boundary violation does result in a new value that is larger than the original one.

However, given the structure of the noise, the exploration of the state-space of the probability is much slower toward the left than toward the right. Having then a bias toward smaller values is not as an important issue as one may think at first. Because of the bouncing, the second method is most likely suited only when the noise distribution is symmetric, which is not the case here. Henceforth, we will use the first method to deal with the boundary problem, with a predefined (high) limit of trials in order to avoid the system to get trapped on a long sequence of unsuccessful iterations. In case the limit of trials is exceeded, the third method will be applied.

4.1.3 Message Structures and Communications

Two nodes are able to exchange messages when they get within mutual communication range. Once it happens, they perform the following steps:

1. exchange node IDs;
2. exchange header information of data messages;
3. each node decides which messages should be forwarded to the other node;
4. messages are exchanged;
5. each node can drop some messages from its buffer (if full) in order to free space for new messages.

The evolving protocol makes use of two types of messages to be exchanged over the network: DATA messages and ACK messages. DATA messages are those carrying the payload transmitted by any mobile node to a specific destination, whereas ACK messages are used for the following purposes:

- to acknowledge the successful delivery of the message at its intended destination;
- to feed back the reward to the nodes along the successful path from source to destination (rewarding);
- to serve as anti-DATA, by blocking the diffusion of already delivered messages and removing them from nodes buffer.

The fields common to all message headers are (i) [message ID], which is the (unique) identifier (ID) of each message and also specifies whether it is a DATA or an ACK message (ii) [GenTime], which describes the time at which the message has been generated. Further, data messages include a hop-count field [hops] and the labels of all nodes which forwarded the message along the path from the source to the actual node. ACK messages, on the other hand, include the complete set of nodes involved in the forwarding path and the DATA message delivery time T_D . Each mobile node maintains two internal data structures dedicated to the storage of DATA and ACK messages respectively. In the structure storing DATA messages, each item additionally stores a counter of the number of copies of that message already disseminated in the network. For any DATA message to be relayed, a node first adds its own node ID to the header and then increments by one the [hops] field of the message. The message is kept in the node internal memory until the corresponding ACK message is received or its lifetime expires.

In addition to DATA messages, mobile nodes diffuse also ACK messages. Unlike the case of DATA messages, no limiting policy is applied to the forwarding of these messages (ACK messages are simply *flooded* into the network according to the VACCINE recovery scheme [14]). Whenever it receives an ACK message, a node first adds the received ACK message to the internal message list. It then checks whether the corresponding DATA message is present in its internal memory and, in case, removes it. If the node has contributed to the successful path to the destination, it updates its own fitness, as described in the previous section.

The overall procedure is summarized in Algorithm 1.

Algorithm 1 Algorithm performed by a node upon reception of an ACK message.

```

1: Add the received ACK message to the internal ACK messages list
2: if msgID  $\in$  {msgID 1, ..., msgID L} then
3:   Remove the corresponding DATA message from the internal structure.
4:   if Node ID  $\in$  W then
5:     Update node's fitness value (REWARDING).
6:   end if
7: end if

```

4.2 Performance Evaluation

In order to evaluate the performance of the presented algorithms, we have run extensive simulations using the freely available simulation tool OMNeT++ [9].

We consider N mobile nodes, moving at constant speed v over a $L \times L$ square playground according to the random direction mobility model [11]. Each node selects the angular direction of its next movement uniformly in $[0, \pi]$, moves along this direction with a uniform speed; upon reaching the border, it generates a new angular direction and moves accordingly. Nodes initial locations are sampled from a uniform distribution which is the nodes stationary distribution under this mobility model (perfect simulation).

Distinct nodes are considered to be in communication range if the mutual distance falls below the communication range r . Each mobile node generates a DATA message in a time interval which is uniformly distributed between 0 and T_s seconds, with a destination chosen uniformly among the nodes in the simulation. Each message generated is stored in the out queue of the generating node. The position of every mobile node in the simulation is updated every T_{step} seconds. Each generation lasts for T_g units of time. The specific values used are in Table 2. The default values of the mutation process are $p_m = 0.01$ and $b = 0.25$ ($a = b\sqrt{2}$, $c = 3/(b^3(3 + 2\sqrt{2}))$); see Fig. 2).

4.2.1 System Convergence

Our first objective is to assess the ability of the evolutionary forwarding mechanism to reach the optimal operating point, which is represented by the minimum of the cost function across all the possible values of the forwarding probability.

Table 2 Simulation parameters

$L = 2500$ m	$T_g = 1500000$ s	$\gamma = 200, 1000, 1800$ s
$r = 25$ m	$T_s = 10000$ s	$p_m = 0.1, 0.2$
$v = 1$ m/s	$T_{\text{step}} = 2$ s	$b = 0.25, 0.5$
Static scenario	$N = 20, 100, 500$	
Dynamic scenario	N varies in $\{20, 60, 80\}$	

We have thus conducted a series of simulation runs with $N = 100$ nodes in which message delivery was achieved through pure probabilistic forwarding, with all nodes applying the same fixed forwarding probability. We ran simulations varying the forwarding probability P from 0 to 1 and computed the cost for $\gamma \in \{200, 1000, 1800\}$. For each γ we conducted simulations using our evolutionary forwarding scheme. Nodes genotypes are initialized by selecting forwarding probabilities uniformly at random in the interval $[0, 1]$.

In Fig. 3 each curve shows the cost, as expressed in (2), achieved by the probabilistic forwarding scheme as the forwarding probability changes, for a specific value of the parameter γ . As one could expect, for low values of γ , like $\gamma = 200$ (corresponding, roughly speaking, to a scenario where resources are not an issue but low delays are required), the cost function is monotonically decreasing in P and flooding ($P = 1$) is the best forwarding policy. As γ increases, the number of message copies done in the system becomes increasingly important, and naturally, smaller forwarding probabilities start achieving lower cost values. For $\gamma = 1000$ and 1800, a minimum exists and a trade-off between low delay and low resource consumption can be found. Three dots represent the performance of our scheme after the initial transient: the abscissas are obtained averaging forwarding probability values across all the nodes. As it can be easily seen, the system running our scheme is able to reach, after convergence, an operating point that is very close to the optimal one, for the three values of γ that were considered.

In the proposed system, evolution occurs at regular intervals, the generation period. Hence, at each generation instant an evolution occurs, and the system changes its behavior updating the forwarding probability toward those values minimizing the cost function. This effect is evident in Fig. 4, which depicts the distribution of the forwarding probability and the cost function CF over the generation number. As it might be seen, after approximately 20 generations, the forwarding probabilities used in the system converge. The average forwarding probability values after

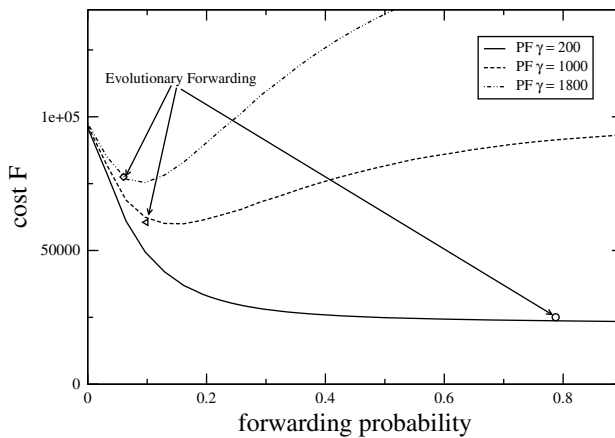


Fig. 3 Cost with Probabilistic Forwarding (PF) and evolutionary forwarding

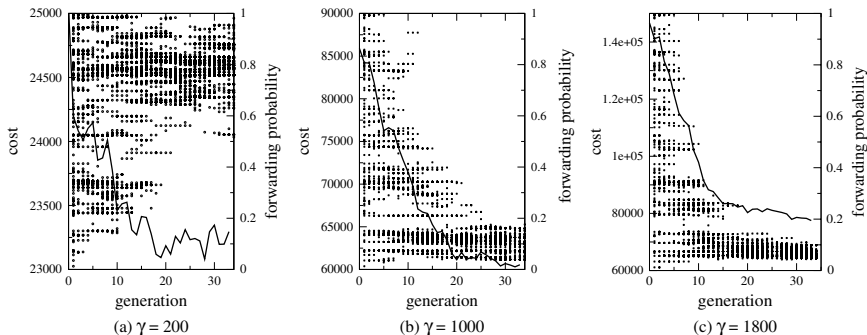


Fig. 4 Cost function and genotypes evolution over time when $N = 100$ nodes

convergence are the ones that are reported on each curve of Fig. 3. The system does converge around the optimal operating point.

Impact of the Mutation Process

We have then analyzed the impact of different evolution parameters, i.e., mutation probability, noise settings. In Fig. 5, we report the distribution of the forwarding probability, together with the cost function, over generations. Figure 5(a) reports the performance under our default mutation settings. In Fig. 5(b), we have kept the same distribution of noise (process Y in Sect. 4) but have considered a larger mutation probability $p_m = 0.2$. The convergence of the cost function appears to be a couple of generations faster. In Fig. 5(c), we have kept $p_m = 0.1$ but have considered a process Y with a larger variance (fourfold the one of the default settings). Interestingly, the system converges much faster than with the default settings for either $p_m = 0.1$ or 0.2 . The convergence seems to occur around the 10th generation. Also, the variance of the noise has a larger impact on the convergence than the probability of mutation.

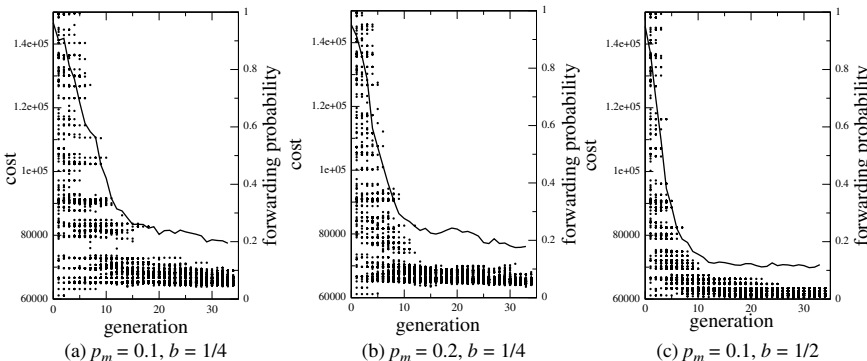


Fig. 5 Impact of the mutation process ($N = 100$ nodes, $\gamma = 1800$)

4.2.2 System Scalability

We have next considered the scaling properties of the proposed evolutionary mechanism. We repeated the same steps described in Sect. 4.2.1 and whose results are reported in Fig. 3, but this time with $N = 20$ and 500. The results are reported in Fig. 6. Figure 6(a) (resp. 6(b)) depicts the cost function versus the forwarding probability when a fixed probabilistic forwarding is used, with $N = 20$ nodes (resp. $N = 500$ nodes). The operating point to which our evolutionary scheme converges is also reported. We can say that the system, when running our scheme, reaches an operating point that is close to the optimal one for a large range of network sizes (N ranging from 20 to 500). We believe that our scheme should scale to even larger network sizes. Unfortunately, fine-grained simulations do not scale as easily to large networks.

4.2.3 System Robustness

The last point that we wanted to address through simulations concerns the robustness of our scheme and its ability to adapt to a changing environment. Therefore, we considered a scenario in which the number of nodes changes abruptly after 20 generations from 10 nodes, up to 100 nodes, then up to 300 nodes after other 20 generations. It then reduces to 100 at the 80th generation and finally reduces back to 10 nodes at the 100th generation. The time-equivalent cost of a copy is set to $\gamma = 1800$ and the noise parameter b is set to 0.5.

Figure 7 depicts, over the generations, the cost achieved by our scheme and the average forwarding probability among all nodes. The cost plot presents spikes whenever N increases. The abrupt change is mainly due to the arrival of new nodes, whose initial forwarding probability is set uniformly at random. During the transient following the spike, genotypes better fitted to the new scenario are identified and the cost reduces. As an example, the system is able to let the forwarding probability decrease when the number of nodes increases. However, a similar behavior is not observed in the opposite case—when the number of nodes decreases. This is rooted in the limited *diversity* of the nodes' genotypes. When adding nodes, the newcomers

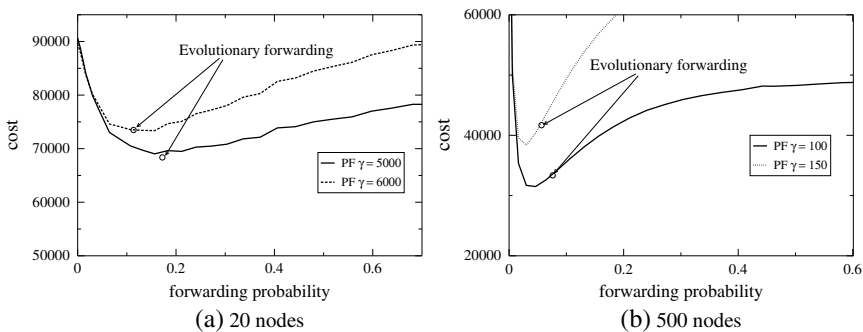


Fig. 6 Convergence of the evolutionary forwarding scheme in the case of 20 and 500 nodes

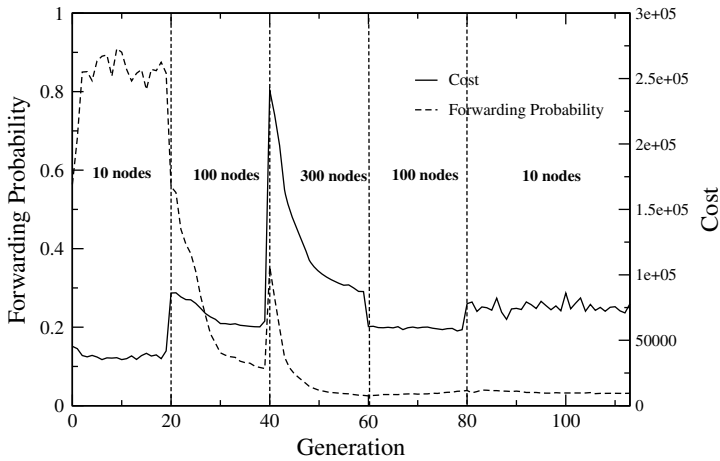


Fig. 7 Dynamic scenario: cost function and forwarding probabilities over the generations

take a forwarding probability chosen at random. This randomness injects a sufficient level of genotype diversity that allows our scheme to explore, in parallel, a wide range of forwarding probabilities. Conversely, once the system has converged to the optimal value for a specific setting and the scenario changes, the system is extremely slow in its reaction due to a limited genetic diversity. This is evidently clear in Fig. 7, where our scheme is not able to track the changes deriving from the removal of nodes.

4.3 Alternative Approaches

In our scheme we rely on standard EC techniques to select genotypes from the pool at each node and to generate new ones. In reality, as Fig. 3 shows, the fitness landscape in our networking problem is quite regular, hence we can think to take advantage of this.

A possibility is to use gradient descent-like techniques. The genotype–fitness pairs in the pool of a node can be considered as samples of the whole fitness landscape. These samples can be used to estimate the gradient of the fitness curve in correspondence of the genotype used by the node and then change it moving *nearer* to the minimum. Fitness samples are noisy, hence the gradient has to be carefully evaluated. At the same time taking into account the intrinsic correlation of the fitness values for similar genotypes should reduce the effect of noise: e.g., an anomalous fitness estimate could lead many nodes to erroneously select that genotype if standard biased selection is applied, but it should not alter significantly gradient calculations.

This idea can be further developed. In fact in our scenario, a node is not constrained to a local gradient evaluation which relies only on fitness estimates for *near* genotypes, but it can exploit all the genotypes in its pool, which are in general

spread across all the genotype space. The minimum of the function can be directly estimated using all the fitness values, hence further reducing the effect of noise. For example, a least-squares fitting can be applied, the minimum can then be evaluated on the basis of the estimated curve, and the node genotype can be made closer to this minimum. This approach should also be integrated with a random mutation, otherwise after some time all genotypes would converge to the same value.

Another possible approach is to renounce to genotype exchange and let each node learn autonomously relying only on its past experience. The same mechanisms described up to now can be applied if we think about the pool as a collection of genotypes and related fitnesses as evaluated in the past by the node itself (and not by different nodes). This solution would definitely make the system more robust to malfunctioning or malicious nodes, which can currently influence genotype selection by other nodes, just by spreading false fitness values, but at the same time longer convergence time would be expected.

5 Conclusion

In this chapter, we have presented a framework to learn in a distributed and online way, a good forwarding policy in DTNs. The most challenging aspect of this framework concerns the estimation of the fitness of the genotype used at a node. Each node contributes to maximizing a global objective function using local knowledge. We have proposed a case study to illustrate the framework. Extensive simulations have illustrated (i) the convergence of the system even when the network counts as much as 500 nodes, (ii) the impact of the mutation process onto the convergence of the system, and (iii) the capability of the system to adapt to changing network conditions.

References

1. Alouf S, Carreras I, Miorandi D, Neglia G (2007) Embedding evolution in epidemic-style forwarding. In: Proc. of IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2007), Pisa, Italy
2. Babaoglu O, Canright G, Deutsch A, Caro GAD, Ducatelle F, Gambardella LM, Ganguly N, Jelasity M, Montemanni R, Montresor A, Urnes T (2006) Design patterns from biology for distributed computing. *ACM Trans Auton Adapt Syst* 1(1):26–66
3. Fall K (2003) A delay-tolerant network architecture for challenged Internets. In: Proc. of ACM SIGCOMM 2003, ACM, New York, USA, pp 27–34
4. Groenevelt R, Nain P, Koole G (2005) The message delay in mobile ad hoc networks. *Performance Evaluation* 62(1–4):210–228
5. Grossglauser M, Tse D (2002) Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Trans on Networking* 10(4):477–486
6. Haas ZJ, Small T (2006) A new networking model for biological applications of ad hoc sensor networks. *IEEE/ACM Trans on Networking* 14(1):27–40
7. Lindgren A, Doria A, Schelen O (2004) Probabilistic routing in intermittently connected networks. In: Proc of SAPIR Workshop 2004, LNCS, vol 3126, pp 239–254

8. Neglia G, Zhang X (2006) Optimal delay-power tradeoff in sparse delay tolerant networks: a preliminary study. In: Proc of ACM SIGCOMM CHANTS 2006, pp 237–244
9. OMNeT (2007) OMNeT++ Discrete Event Simulation System. <http://www.omnetpp.org>
10. Pelusi L, Passarella A, Conti M (2006) Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Comm Mag* 44(11):134–141
11. Royer EM, Melliar-Smith PM, Moser LE (2001) An analysis of the optimum node density for ad hoc mobile networks. In: Proc. of IEEE ICC 2001, vol 3, pp 857–861
12. Spyropoulos T, Psounis K, Raghavendra CS (2008) Efficient routing in intermittently connected mobile networks: The multiple-copy case. *IEEE/ACM Trans on Networking* 16(1): 77–90
13. Vahdat A, Becker D (2000) Epidemic routing for partially connected ad hoc networks. Tech Rep CS-200006, Duke University
14. Zhang X, Neglia G, Kurose J, Towsley D (2007) Performance modeling of epidemic routing. *Computer Networks* 51(10):2867–2891

Dynamic and Fair Spectrum Access for Autonomous Communications

Jianhua He, Jie Xiang, Yan Zhang, and Zuoyin Tang

Abstract Most of existing wireless systems are regulated by a fixed spectrum assignment strategy. This policy leads to an undesirable situation that some systems may only use the allocated spectrum to a limited extent while others have very serious spectrum insufficiency situation. Dynamic spectrum access (DSA) is emerging as a promising technology to address the above issue such that the unused licensed spectrum can be opportunistically accessed by the unlicensed users. To enable DSA, the unlicensed user shall have the capability of detecting the unoccupied spectrum, controlling its spectrum access in an adaptive manner, and coexisting with other unlicensed users automatically. In this chapter, we will give an overview on the DSA for sharing open spectrum. And we investigate a radio system transmission opportunity (TXOP)-based spectrum access control protocol, with the aim of improving spectrum access fairness and ensure safe coexistence of multiple unlicensed radio systems. Simulation is carried out to evaluate the TXOP-based scheme with respect to spectrum utilization, fairness, and scalability.

1 Introduction

Recent advances in information technologies have boosted broadband multimedia services. It is envisioned that broadband multimedia services will be increasingly popular over a variety of wireless mobile networks. With the increasing traffic load, wireless radio spectrum is becoming more congested and expensive. Traditionally, spectrum management agency licenses a fixed frequency band to the wireless service operators. This strategy leads to an undesirable situation that some systems may only use the allocated spectrum to a limited extent while others have very serious spectrum insufficiency situation. Recent studies have found that more than 70% of spectrum is unutilized in most areas [1]. The efficient use of the unoccupied spectrum (also referred to as open spectrum) will promote spectrum sharing and ease the spectrum shortage challenge [2, 3]. As a highly promising technology to

J. He (✉)

Institute of Advanced Telecommunications, Swansea University, Swansea SA2 8PP, UK
e-mail: j.he@swansea.ac.uk

address spectrum insufficiency problem, dynamic spectrum access (DSA) is attracting an increasing interest in both research community and industry. DSA is under investigation as part of cognitive radio system to achieve operation in a wide range of frequency bands with high accessibility and utilization [4–6]. In DSA, unlicensed users (or secondary users) dynamically use available spectrum without interfering with licensed users, aiming to increase spectrum utilization and simplify spectrum management.

Although DSA is highly promising, it is still in the very early stage of research and development. There are a number of technical, economical, and regulatory challenges. For instance, it is a fundamental problem for the secondary users to efficiently detect the unoccupied spectrum opportunity by the primary wireless system and not interfere with the normal communications in the primary wireless system. Due to limitation or unavailability of central control entity, it is also a big challenge faced by DSA to adaptively control its spectrum access and automatically coexist with other secondary users. With multiple different types of radio systems in sharing the opportunistic spectrum in a distributed manner, it becomes complicated to provide fairness in sharing the opportunistic spectrum across users and minimize communications overhead while simultaneously achieving efficient spectrum usage.

This chapter is structured as follows. In Sect. 2, we overview the classification of current DSA schemes, and explore the DSA technologies in three scenarios, i.e., secondary users (SUs) coexistent with primary users (PUs), SUs in the same radio system, and SUs in different radio systems. Section 3 introduces a channel access model in open spectrum wireless networks. Section 4 describes the details of three DSA protocols. Section 5 presents simulation results and numerical examples. Finally, conclusions are given in Sect. 5.

2 Overview of Dynamic Spectrum Access Schemes

There are two classification results in the literature. The first one is presented in [7], where the authors classify DSA from three different aspects, i.e., architecture, spectrum allocation behavior, and spectrum access technique. From the architecture's point of view, DSA can be classified into centralized and distributed manners. In spectrum allocation behavior, both cooperative and non-cooperative behaviors are analyzed. In spectrum access technique, SUs can work with PUs in two ways. One is called underlay mode, where SUs can work on all of the channels if the interference to the PUs is under a predefined threshold. The other one is called overlay mode, where SUs can only work on the channels which are not occupied by the PUs.

Another classification is presented in [8], where the authors classify the DSA into three models, i.e., Dynamic Exclusive Use Model, Open Sharing Model, and Hierarchical Access Model. In Dynamic Exclusive Use Model, SUs and PUs work in different spectrum bands by the following strategies: regulation, dynamical allocation, etc. According to this model, several research works suppose that licensed spectrum

regulator can sell the spectrum bands to unlicensed users. So that SUs can access the spectrum different with PUs, after they buy the spectrum bands. Recent research works extend this scenario to multisellers of different spectrum bands, where they will compete with each other to attract more SUs as their customers [9]. The Open Sharing Model treat the spectrum resource as the common property which can be used equally to the different users in the network. This model is currently used for the unlicensed spectrum bands such as the industry, science, and medical (ISM) bands. In Hierarchical Access Model, the authors also classify it into underlay and overlay mode which is the same with the hierarchical access model in [7].

2.1 Dynamic Spectrum Access for SUs Coexistent With PUs

How do SUs access the spectrum bands coexistent with PUs is a fundamental problem in Cognitive Radio Networks. Figure 1 shows the concept of DSA for SUs coexistent with PUs. In the literature, several schemes have been studied with the underlay and overlay approach, as summarized in [7] and [8].

In the underlay approach, every SU will control its transmission power in order to guarantee that the interference to PU is limited to the threshold. The key issue for the underlay approach is how to measure the interference temperature on PUs in an efficient way [10]. Several works have been done by considering the interference temperature constraints for SUs access the spectrum bands, such as [11] and [12].

In the overlay approach (also called opportunistic spectrum access), SUs detect the appearance of PUs in real time, and use the spectrum bands which are not occupied by any PU. In this case, PU detection (also known as spectrum detection or spectrum sensing) is the significant issue. In the literature, there are four

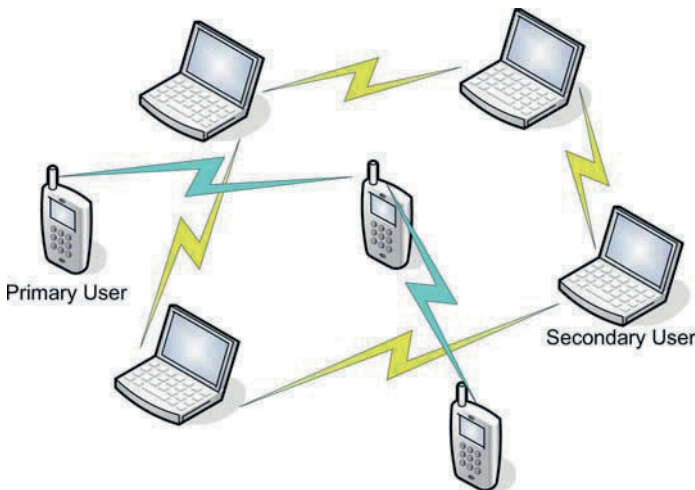


Fig. 1 Concept of dynamic spectrum access for SUs coexistent with PUs

major methods for spectrum sensing, i.e., Matched filter, Energy detection [13], Cyclostationary detection [14], and Wavelet detection [15]. Each method has its own advantages and disadvantages to different scenarios. Detecting the event of PU transmission by a single node is not effective when the SU is shadowed from the PU, or when the SU is out of the PU’s transmission range but it can still interfere with the primary receiver inside the PU’s transmission range [7]. Therefore, cooperative sensing [16, 17], which allows several nodes sense the spectrum environment and make the decision in a cooperative manner, is thought to be an efficient way to solve such problems.

Several DSA schemes [18–22] are proposed using the sensing-based opportunistic spectrum access approach. For instance, in [18], SUs utilize the past observations to build predictive models for spectrum availability, and choose the channels with the most availability metric. In [19], the authors suppose that SUs can only sense some of the available channels because of hardware and energy constraints, and derive the spectrum access strategies under the formulation of finite-horizon partially observable Markov decision processes. In [22], the authors extend the work in [19]. They model the channel occupancy by PUs with a continuous-time Markov chain, and propose an opportunistic spectrum access scheme via periodic channel sensing, while reducing the complexity of the optimal solution in [19].

2.2 Dynamic Spectrum Access for SUs in the Same Radio System

After obtaining the spectrum opportunity from primary systems, how to share the spectrum between SUs working in the same radio system in a fair and efficient way is another challenging problem. Figure 2 shows the concept of DSA for SUs in the same radio system. Every link between two SUs will work on one spectrum band from the available channels.

In [23], a protocol for coordinated spectrum access called Dynamic Spectrum Access Protocol (DSAP) is proposed. The authors suppose that there is

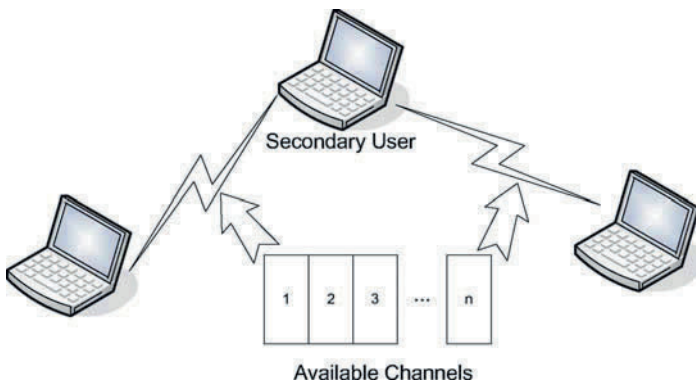


Fig. 2 Concept of dynamic spectrum access for SUs in the same radio system

a coordinating central entity called DSAP server who allot the spectrum resource in the similar manner as the Dynamic Host Configuration Protocol (DHCP) server allots IP address to hosts in a network.

Distributed methods are also introduced by several researchers. For instance, in [24], the authors propose a distributed rule-regulated spectrum sharing scheme, where nodes share the spectrum resource fairly by making independent actions following spectrum rules. The rule-based approach significantly reduces the communication cost comparing to the explicit coordination approach, while achieving similar performance.

2.3 Dynamic Spectrum Access for SUs in Different Radio Systems

Since different radio systems cannot communicate with each other, it becomes more complicated to provide fairness for SUs from different radio systems in sharing the opportunistic spectrum across users and minimize communications overhead while simultaneously achieving efficient spectrum usage. Figure 3 shows the concept of DSA for SUs in different radio systems. Several works have been done on this problem, such as [2, 25, 26].

In [25], the authors solve the problem of the DSA between two different radio systems, i.e., IEEE 802.11b and IEEE 802.16a. Two methods are used in their work. The first one uses reactive interference avoidance algorithms, where radio nodes coordinate spectrum usage without exchange of explicit control information. However, the hidden-receiver problem emerges in this method. The second one employs a common spectrum coordination channel to exchange of the transmitter

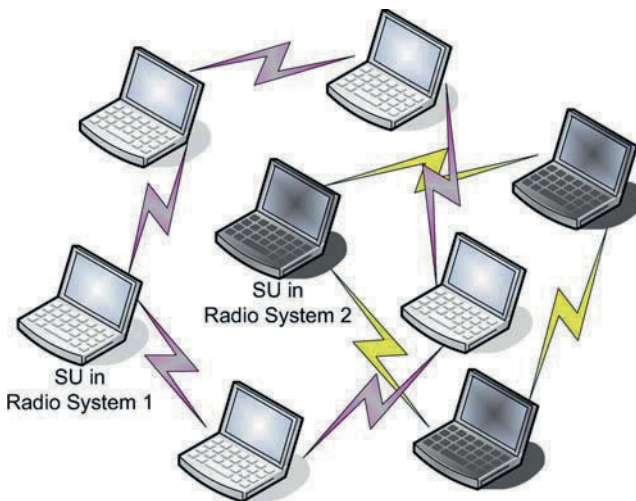


Fig. 3 Concept of dynamic spectrum access for SUs in different radio systems

and receiver parameters following the spectrum etiquette policies, and can solve the hidden-receiver problem.

In [26], the authors employ the game theory, especially using a repeated game, to model the interaction between different radio systems. They can obtain the fairness and efficiency of access the spectrum bands when the game achieve the equilibrium. However, how to reduce the computation complexity and the time required on the game convergency is still a key issue for employing game theory to solve DSA problems.

Specially, two dynamic DSA protocols have been proposed by [2]. The first one aims to optimize spectrum access fairness, which is relying on the centrally optimized channel access probabilities for each radio systems. The channel access probabilities are obtained via an analytical model. The centralized optimization requires the system detailed information (e.g., protocols specification, number of terminals, and traffic statistics in each radio system) in order to model the spectrum access protocols and calculate optimal transmission probabilities. Therefore, it may achieve the optimal fairness in theory for the specific protocols. This may become impractical and inefficient for the heterogeneous wireless systems, which operate dynamically and adaptively in a distributed approach. The second protocol is proposed to overcome the drawbacks of the first protocol by dynamically adjusting channel access probability with local measurements.

The second protocol is implemented in a distributed way. High fairness on sharing opportunistic spectrum is the main target. However, the second protocol requires that each radio system should accurately measure the channel access durations of each competing radio system and has the traffic statistics of each radio system. In practice, it will be difficult for a radio system to real-time monitor the channel access time of other radio systems. It is also very likely that the algorithm may take a long time to converge, which is undesirable in an opportunistic open spectrum wireless network since the available spectrum may change dynamically and quickly depending on time and location.

In this chapter, we evaluate the above two DSA protocols for open spectrum wireless networks. We analyze the spectrum access problems and propose a new radio system transmission opportunity (TXOP)-based spectrum access control protocol. In our proposed scheme, each radio system will obtain a TXOP by competition or by assignment from a network coordinator. During a beacon period, each radio system monitors its channel usage. If a radio system uses the channel longer than the TXOP in a beacon, it will stop transmitting packets in the rest of the beacon period. As each radio system needs only monitor the channel activity of the users from its own system and this can be done by a member of this radio system, it eliminates the strong requirement of inter-radio system communications. It also does not require the information of the protocols used for channel access and traffic characteristics. Thereafter, better scalability and higher flexibility can be achieved. In the next sections, we will present the TXOP-based scheme and compare the spectrum access protocols in terms of spectrum utilization, fairness in sharing open spectrum, and scalability.

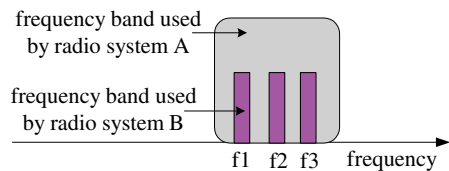
3 Channel Access Model

As we are interested in general DSA, an abstract channel access model is used as shown in Fig. 4. For the sake of illustration, we assume two different types of radio systems (A and B) operating in the band. Each radio system has N_i ($i \in \{A, B\}$) active user terminals. It is noteworthy that the model can be easily extended to the multiple radio systems. The type A radio system represents a wideband system operating on the frequency channel centered on f_2 . The type B radio system represents a narrowband system operating on the three frequency channels (center frequencies f_1, f_2, f_3). The type A frequency channel overlaps with all the three type B frequency channels. This model is a simplified version of that used by Xing et al. [2], in which three sets of three narrowband frequency channels are considered while here only one set of three narrowband frequency channels is studied. The listen-before-talk (LBT) strategy is applied for all radio systems to avoid modeling the detailed protocols by Xing et al. [2]. In this paper, we used Carrier Sense Multiple Access (CSMA) protocol for channel access, which will be introduced in the next subsection. CSMA will be adjusted for DSA. We assume that the traffic arrival process follows Poisson process with the arrival rate λ_i in radio system i . Hence, the packet inter-arrival time is exponentially distributed with mean time $1/\lambda_i$ second. The radio system access duration is constant with T_i for radio system i with $T_i = Rate_i/PLe_n_i$, where $Rate_i$ and PLe_n_i represent transmission rate and packet length in radio system i .

3.1 Basic Carrier Sense Multiple Access Protocol

Before proceeding on, we have an introduction of the basic CSMA protocol, which is the baseline of the DSA protocols used by Harada and Prasad [27]. Multiple access protocol is defined as the agreement and set of rules among users for the successful transmission of information over a common medium. The multiple access protocols can be classified into three major categories: contention-free protocols, contention-based protocols, and CSMA protocols. In contention-based protocols, there is no common scheduling of transmissions and hence devices will randomly access the channel. CSMA is a contention-based multiple access protocol. In the CSMA protocol, each access terminal needs to judge if the communication channel is used by other terminals before packet transmission by the means of sensing the existence of the carrier wave on the communication channel. If a carrier wave is sensed on the communication channel, the channel is thought to be busy and the terminal will not transmit packet in order to avoid collision. In this paper, we focus

Fig. 4 A example of frequency channels used by two types of radio systems (A, B). Each radio system represent a group of communication radio devices



on the nonpersistent CSMA protocol. In the nonpersistent CSMA protocol, when packets are generated in a terminal, the terminal starts the carrier sensing. If the channel is sensed idle, the terminal sends the packet to the destination immediately. Otherwise, the terminal will hold the packet transmission and wait for a random period to start sensing the channel again. Under an ideal communication environment with infinite call source and no hidden terminals, the theoretic throughput S of nonpersistent CSMA protocol can be express as [27]

$$S = \frac{Ge^{-\alpha G}}{G(1 + 2\alpha) + e^{-\alpha G}} \quad (1)$$

where G denotes the offered traffic load and α represents the normalized propagation delay to packet transmission duration. In the remainder of the chapter, if not specified, CSMA protocol refers to nonpersistent CSMA protocol.

4 Open Spectrum-Sharing Protocols

In this section, we will present three DSA protocols. The main purpose is the efficient coexistence of multiple radio systems, which may operate on different frequency bands. In particular, we will concentrate on the fairness guarantee in spectrum sharing while maintaining high overall spectrum utilization. With respect to fairness, it can be either system-oriented radio or single user-oriented radio. For instance, there are n users contending system resources and the i th user receives an allocation x_i . Let $f(x)$ denote the fairness index [28]. $f(x)$ is given by

$$f(x) = \frac{|\sum_{i=1}^n x_i|^2}{\sum_{i=1}^n x_i^2}, \quad x_i \geq 0. \quad (2)$$

Several other fairness measures are also discussed by Jain et al. [28].

Without any transmission control, some radio systems may dominate the spectrum usage due to large amount of traffic and high density of terminals in the radio systems. Based on the CSMA protocol, we can observe that the channel access behavior can be controlled from three possible ways to differentiate throughput and fairness during the spectrum sharing.

- Control of transmission probability, which means a transmission probability (denoted by ρ_i) is used to randomly control the transmission of a packet after a terminal obtains the right to transmit the packet over the channel.
- Control of waiting time, which randomly controls the duration of waiting time after a terminal detects a busy channel. This is similar to the control of transmission probability method above, and will not discuss in details in this paper.
- Control of transmission opportunity, which is used either in deterministic or random way to control the packet transmission of a terminal/radio system over a period based on a transmission opportunity. This is in contrast to the random

control of each packet transmission by the means of transmission probability control. This control method will be investigated in this paper.

4.1 CSMA With Transmission Probability Control

The two protocols proposed by Xing et al. [2] are based on the control of transmission probability in the simple LBT protocol to achieve fairness in either centralized or distributed manner. Although the idea of controlling transmission probability is not new, the authors proposed a general Markov chain model, which can be helpful on performance evaluation/optimization and long-term network planning. However, this centralized optimization model highly depends on the analytical model and requires accurate whole network information, e.g., the number of stations and traffic loads. This requirement may not match with the practice. The original analytical model is developed for the simple LBT protocol and can be extended to the CSMA protocol.

4.2 CSMA With Dynamic Transmission Probability

As the above centralized transmission probability based control protocol is heavily affected by the network assumption, it is not efficient with respect to scalability and flexibility to the dynamic opportunistic spectrum. It is also difficult to extend the analytical model to multihop networks. This indicates that the proposed model can be only applied in single hop infrastructure-based wireless networks. Motivated by this, a distributed channel access protocol based on anthropological model is proposed in [2] with the support of local information and limited global knowledge. A homo equalis (HE) society model is used. We call this scheme as CSMA-HE protocol,

In an HE society, no centralized governance exists and the enforcement of norms depends on the voluntary participation of peers. An HE society can be modeled with the utility function u_i of player i in an n -player game. The utility function u_i is given by [2]

$$u_i = x_i + \frac{\alpha_i}{n-1} \sum_{x_j > x_i} x_j - x_i + \frac{\beta_i}{n-1} \sum_{x_j < x_i} x_j - x_i \quad (3)$$

where $x = (x_1, \dots, x_n)$ are the payoffs for each player and $0 \leq \beta < \alpha_i \leq 1$.

In the CSMA-HE protocol, each radio system learns the transmission probability p_i by itself. Denote x_i as the average cumulative channel access duration for radio system type of i . With the initial probability $p_i = 1$, the probability p_i is updated following the equation below [2]:

$$p_i = \max(0, \min(1, p_i + \frac{\alpha_i}{n-1} \sum_{x_j > x_i} x_j - x_i + \frac{\beta_i}{n-1} \sum_{x_j < x_i} x_j - x_i)) \quad (4)$$

where n denotes the number of different radio systems, and $0 \leq \beta < \alpha_i \leq 1$. The only local information needed is its own and the other radio systems' histories of cumulative spectrum access duration.

4.3 CSMA With Transmission Opportunity Control

A similar idea of controlling channel access by TXOP has been used in [29]. TXOP is an important concept introduced for quality-of-service (QoS) support for real-time applications in the IEEE 802.11e standard. It is used to control an individual user terminal. In the proposed TXOP-based multiple channel access protocol, TXOP can be used to control the individual user terminals, but our emphasis is to use TXOP to control the channel access of a whole radio system. We denote the TXOP-based CSMA protocol as CSMA-TXOP protocol.

CSMA-TXOP protocol can be described as follows. We assume each radio system operates with beacon signals. Beacon signals need not to be synchronized for different radio systems. The periods of beacon signals can also be different in each radio system. A radio system obtains a TXOP by negotiation/competition with other radio systems in a distributed way or from assignment by a central control entity. The TXOP can be fixed over a long time or change dynamically according to the network conditions. It is to be used in a predefined beacon period. In the beginning of each new beacon period, a radio system resets the measured cumulative channel access duration. Then, it continuously measures its own cumulative channel access duration. If the access duration of a radio system exceeds its TXOP before the arrival of the next beacon signal, the radio system must partially or fully limit its channel access until the next beacon period.

5 Simulation Results

In this section, we will show the illustrative numerical examples and simulation results in the CSMA-P, CSMA-HE, and CSMA-TXOP protocols. CSMA-P protocol represents the CSMA protocol with neither transmission probability control nor TXOP control for packet transmissions. The emphasis of the simulations is to verify the feasibility, effectiveness, and scalability of the CSMA-HE and CSMA-TXOP protocols in single-hop wireless networks. A randomly generated network is shown in Fig. 5, in which there are totally 30 wireless terminals located around an wireless access point, with $N_A = 10$ and $N_B = 20$ for the radio systems A and B, respectively. Wireless access point acts as the control center for the CSMA-TXOP protocol. Simulation results are observed with different configurations. We assume that all the wireless terminals can hear each other and the access point in the single-hop

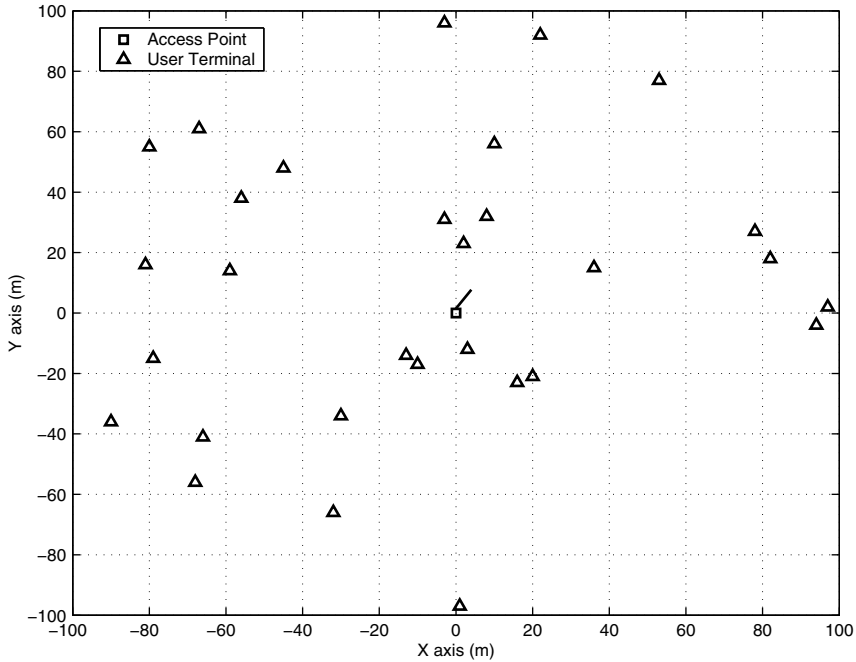


Fig. 5 Network topology. 30 user terminals ($N_A = 10, N_B = 20$) randomly located in $200 \times 200 \text{ m}^2$ area with the access point at the center

networks. Each terminal will access the channel and send traffic to the access point using a channel access protocol. For the sake of simplicity, we assume no packet loss due to low signal-to-noise (SNR). Hidden terminal issues is eliminated by setting propagation delay $d = 0$. Packets are transmitted at 256 kbps. Packet length is fixed as 528 bits.

5.1 Pure CSMA Protocol

Figures 6 and 7 present the throughput of radio system A and B with CSMA-P protocol. Without an external control mechanism for packet transmissions, the two competing radio systems can not fairly share the spectrum. In addition, the spectrum sharing fairness is heavily affected by the traffic load and the number of terminals in each radio system. For example, with $\lambda_A = \lambda_B$, the normalized throughput S_A (S_B) of radio system A (B) is 0.42 (1.28) with the normalized traffic load $G = 4.1$ as shown in Fig. 6. Throughput S_B of radio system B is 3 times of S_A . When the traffic loads satisfy $\lambda_B = 4\lambda_A$, the normalized throughput S_A (S_B) of radio system A (B) is 0.12 (1.38) with $G = 4.1$, as shown in Fig. 7. Throughput S_B is more than 10 times of S_A .

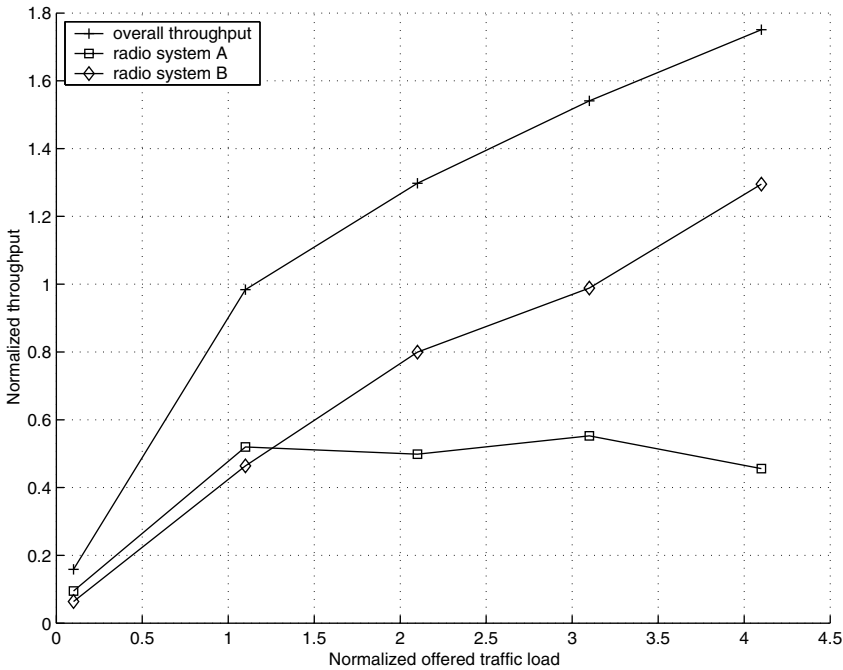


Fig. 6 Normalized throughput versus normalized traffic load with CSMA-P protocol, $\lambda_A = \lambda_B$

5.2 Homo Egulias-Based Channel Access Control

Figures 8, 9, and 10 show the normalized through in case of CSMA-HE protocol with $\lambda_A = \lambda_B$ (solid lines) and $\lambda_B = 4\lambda_A$ (dashed lines). It can be observed that, in both situations, CSMA-HE protocol can achieve better fair spectrum sharing. However, in the condition of $\lambda_B = 4\lambda_A$, the overall spectrum utilization decreases by 40% from 1.6 to 1.0 when the traffic load $G = 4.1$. This indicates that CSMA-HE protocol suffers low spectrum utilization when the traffic loads information of other radio systems is unavailable. Since CSMA-HE protocol is specifically designed to optimize fairness in spectrum sharing, the scheme is difficult to leave the low utilization operating point if the estimated information is inaccurate.

Figures 9 and 10 illustrate the instant transmission probability and instant normalized throughput for $\lambda_A = \lambda_B$ (solid lines) and $\lambda_B = 4\lambda_A$ (dashed line), respectively. The results are able to show the CSMA-HE protocol dynamics over time. Namely, CSMA-HE protocol takes several hundred rounds of adjustment to converge to a relatively stable operation point. As a consequence, the algorithm convergence is slow and infeasible for DSA in heterogeneous wireless networks. It is also noteworthy that the frequent information gathering or exchange will consume extra energy and resources; and hence generates high overhead.

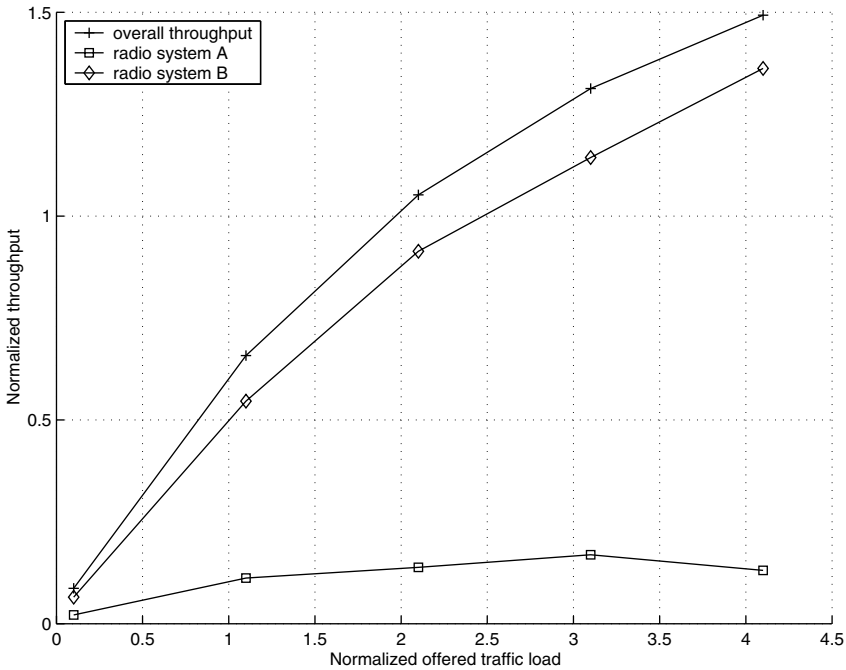


Fig. 7 Normalized throughput versus normalized traffic load with CSMA-P protocol, $4\lambda_A = \lambda_B$

5.3 TXOP-Based Channel Access Control

Figures 11 and 12 show the throughput of the radio system A and B with CSMA-TXOP protocol, respectively. Figure 11 demonstrates the situation when $\lambda_A = \lambda_B$. It is clear that a fair spectrum sharing is achieved between the two radio systems without degrading spectrum utilization. Figure 12 reports the situation when $4\lambda_A = \lambda_B$. Configuration of TXOPs for each radio system can be adjusted to achieve fair spectrum access. On the other hand, the overall spectrum utilization may be degraded a bit. Based on the results, we can improve the spectrum utilization by setting the parameter TXOP to make the system operating away from the optimal fairness point. Though it is difficult to maximize both spectrum utilization and fairness at the same time, CSMA-TXOP protocol is an effective strategy to control and improve system performance based on the instantaneous traffic load, QoS priority, and trade-off between utilization and fairness.

6 Conclusion

Opportunistic spectrum access is a promising technique to address the spectrum shortage problem. In this paper we investigated several DSA protocols for open spectrum shared by heterogeneous wireless networks. The DSA protocols are

designed to enable multiple heterogeneous unlicensed radio systems coexistence and use available spectrum without interfering with licensed users. This will increase spectrum utilization and simplify spectrum management. We propose a new TXOP-based spectrum access protocol. The chief motivation is to ensure coexistence of multiple radio systems and achieve acceptable spectrum utilization and fair spectrum sharing. We evaluate the performances of the proposed scheme and compare with the two existing spectrum access protocols. The simulation result shows that the proposed scheme is able to achieve high scalability and controllability while maintaining spectrum utilization and fairness performance. In the future, we will extend and evaluate the dynamic spectrum sharing protocols in multi-hop wireless networks. It is also an interesting topic to integrate HE and TXOP-based control methods together with multiple access protocols other than CSMA protocol.

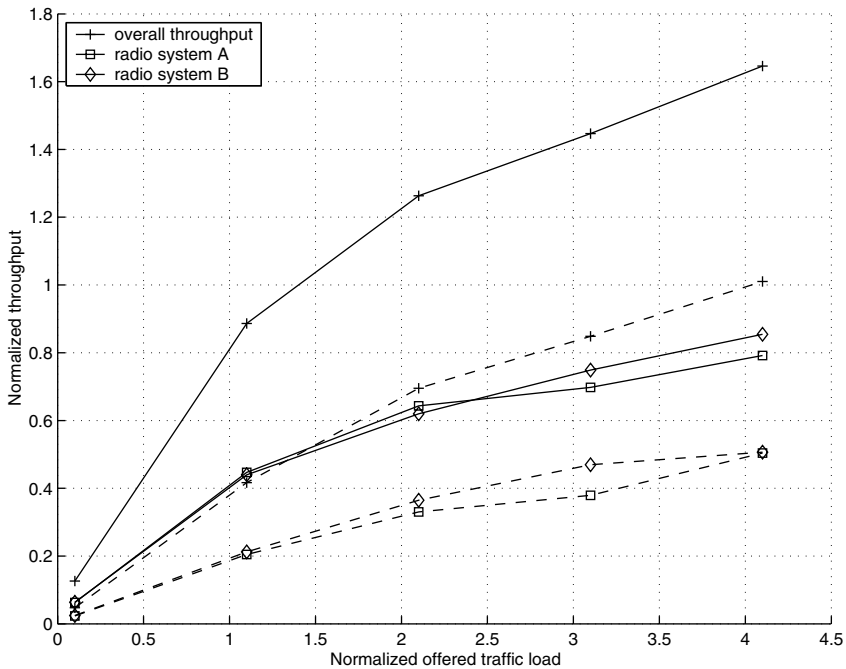


Fig. 8 Normalized throughput versus normalized traffic load with CSMA-HE protocol, $\lambda_A = \lambda_B$ (solid lines) and $4\lambda_A = \lambda_B$ (dashed lines)

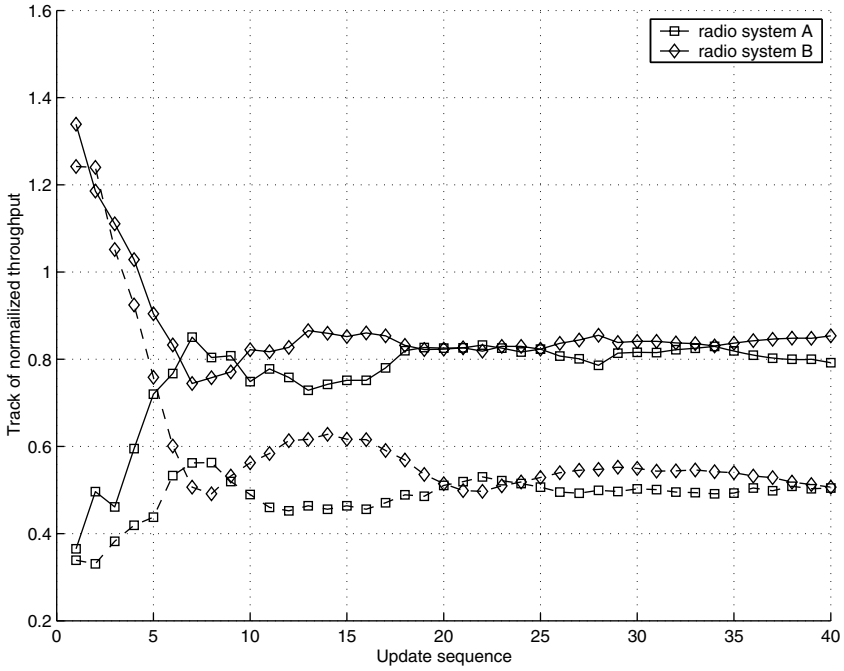


Fig. 9 Instant transmission probability over time with CSMA-HE protocol, $\lambda_A = \lambda_B$ (solid lines) and $4\lambda_A = \lambda_B$ (dashed lines)

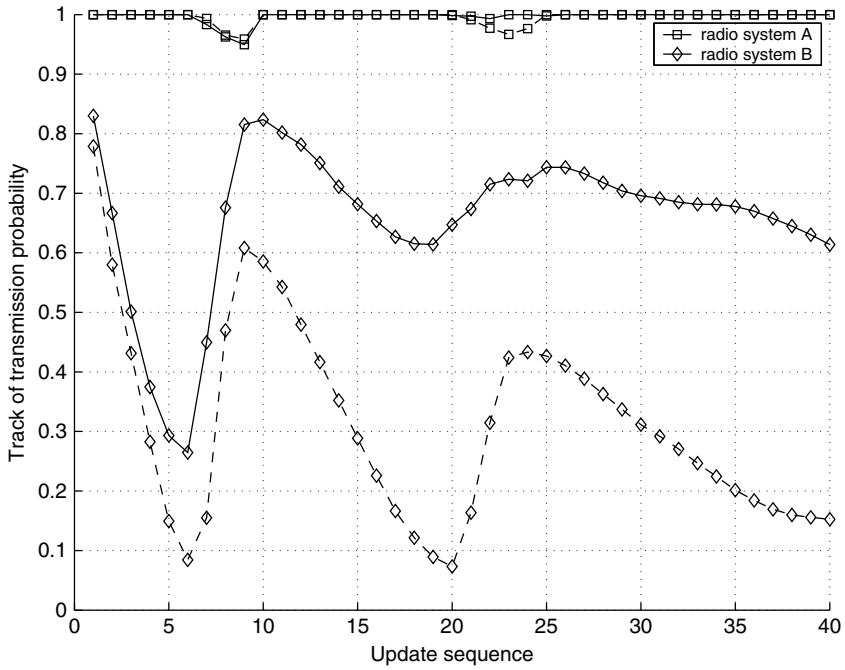


Fig. 10 Instant normalized throughput over time with CSMA-HE protocol, $\lambda_A = \lambda_B$ (solid lines) and $4\lambda_A = \lambda_B$ (dashed lines)

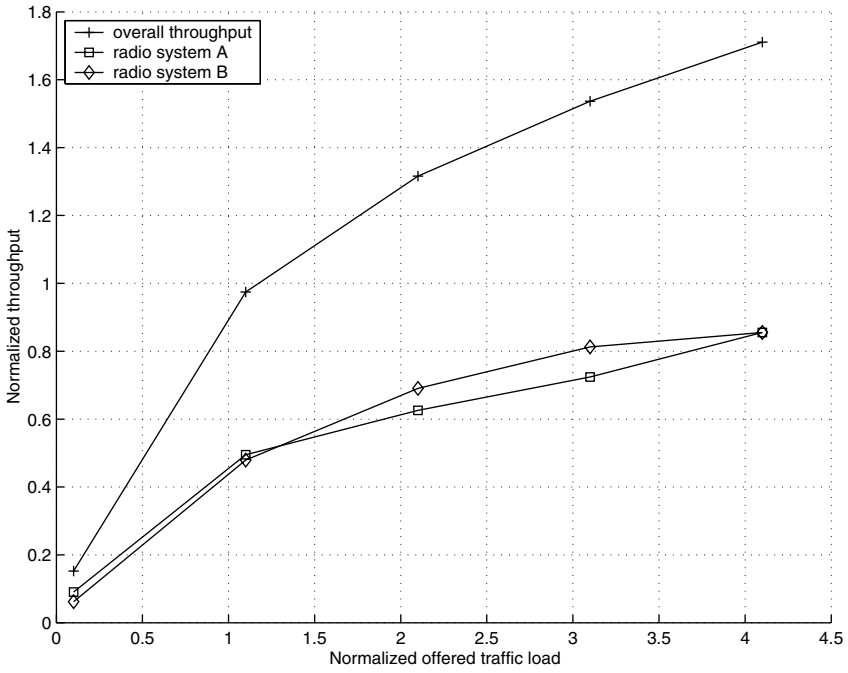


Fig. 11 Normalized throughput versus normalized traffic load with CSMA-TXOP protocol, $\lambda_A = \lambda_B$

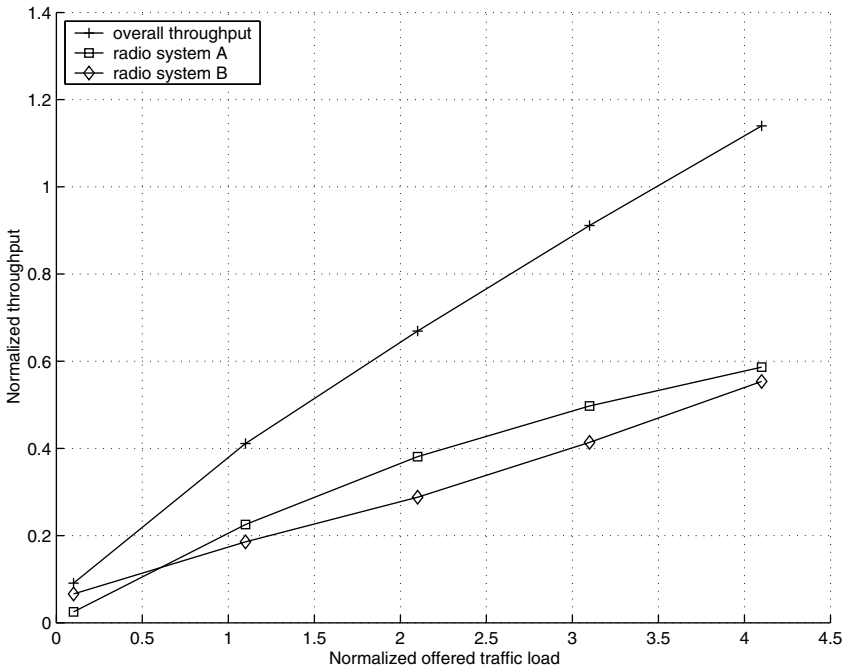


Fig. 12 Normalized throughput versus normalized traffic load with CSMA-TXOP protocol, $4\lambda_A = \lambda_B$

Acknowledgments The work reported in this chapter has been partially funded by the OSIRIS project within the 3C Research programme on convergent technology for digital media processing and communications, and by the European Union through the Welsh Assembly Government.

References

1. XG Working Group. (Jul. 2003) The XG Vision. Request for Comments, BBN Technologies, Cambridge, MA. [Online]. Available: <http://www.darpa.mil/ato/programs/XG/rfcs.htm>
2. Y. Xing, R. Chandramouli, S. Mangold, S. Shankar, "Dynamic spectrum access in open spectrum wireless networks," *IEEE J. Selected Areas Commun.*, Vol. 24, No. 3, March 2006.
3. S. Sankaranarayanan, P. Papadimitratos, A. Mishra, "Bandwidth sharing approach to improve licensed spectrum utilization," *Proc. of IEEE DYSpan2005*, 2005.
4. J. Mitola, "The software radio architecture," *IEEE Commun. Mag.*, Vol. 33, No. 5, pp. 26–38, 1995.
5. S. Haykin, "Cognitive radio: Brain-empowered wireless communications," *IEEE J. Selected Areas Commun.*, Vol. 23, No. 2, pp. 201–220, 2005.
6. D. P. Satapathy, J. M. Peha, "Spectrum sharing without licenses: Opportunities and dangers," *Proc. Telecommun. Policy Res. Conf.*, pp. 15–29, 1996.
7. I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, S. Mohanty, "Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Comput. Netw.*, Vol. 50, No. 13, pp. 2127–2159, 2006.
8. Q. Zhao, A. Swami, "A survey of dynamic spectrum access: Signal processing and networking perspectives," *ICASSP 2007*, Vol. 4, 15–20 April 2007.

9. Y. Xing, R. Chandramouli, C. Cordeiro, "Price dynamics in competitive agile spectrum access markets," *IEEE J. Selected Areas Commun.*, Vol. 25, No. 3, pp. 613–621, April 2007.
10. A. E. Leu, M. McHenry, B. L. Mark, "Modeling and analysis of interference in listen-before-talk spectrum access schemes," *Int. J. Netw. Manag.*, Vol. 16, No. 2, pp. 131–147, 2006.
11. M. Sharma, A. Sahoo, K. D. Nayak, "Channel selection under interference temperature model in multi-hop cognitive mesh networks," *DySPAN 2007*, pp. 133–136, 17–20 April 2007.
12. J. Bater, H.-P. Tan, K. N. Brown, L. Doyle, "Modelling interference temperature constraints for spectrum access in cognitive radio networks," *ICC '07*, pp. 6493–6498, 24–28 June 2007.
13. F. F. Digham, M.-S. Alouini, M. K. Simon, "On the energy detection of unknown signals over fading channels," *IEEE Trans. Commun.*, Vol. 55, No. 1, pp. 21–24, Jan. 2007.
14. P. Sutton, K. Nolan, L. Doyle, "Cyclostationary signatures in practical cognitive radio applications," *IEEE J. Selected Areas Commun.*, Vol. 26, No. 1, pp. 13–24, Jan. 2008.
15. Y. Youn, H. Jeon, H. Jung, H. Lee, "Discrete wavelet packet transform based energy detector for cognitive radios," *VTC2007-Spring*, pp. 2641–2645, 22–25 April 2007.
16. A. Ghasemi, E. Sousa, "Collaborative spectrum sensing for opportunistic access in fading environments," *DySPAN 2005*, pp. 131–136, 8–11 Nov. 2005.
17. M. Gandetto, C. Regazzoni, "Spectrum sensing: A distributed approach for cognitive terminals," *IEEE J. Selected Areas Commun.*, Vol. 25, No. 3, pp. 546–557, April 2007.
18. P. A. K. Acharya, S. Singh, H. Zheng, "Reliable open spectrum communications through proactive spectrum access," in *TAPAS '06: Proceedings of the First International Workshop on Technology and Policy for Accessing Spectrum*. New York, USA, ACM, 2006, p. 5.
19. Q. Zhao, L. Tong, A. Swami, Y. Chen, "Decentralized cognitive mac for opportunistic spectrum access in ad hoc networks: A pomdp framework," *IEEE J. Selected Areas Commun.*, Vol. 25, No. 3, pp. 589–600, April 2007.
20. S. Geirhofer, L. Tong, B. Sadler, "Cognitive radios for dynamic spectrum access – dynamic spectrum access in the time domain: Modeling and exploiting white space," *IEEE Commun. Mag.*, Vol. 45, No. 5, pp. 66–72, May 2007.
21. K. Chowdhury, I. Akyildiz, "Cognitive wireless mesh networks with dynamic spectrum access," *IEEE J. Selected Areas Commun.*, Vol. 26, No. 1, pp. 168–181, Jan. 2008.
22. Q. Zhao, S. Geirhofer, L. Tong, B. Sadler, "Opportunistic spectrum access via periodic channel sensing," *IEEE Trans. Signal Process.*, Vol. 56, No. 2, pp. 785–796, Feb. 2008.
23. V. Brik, E. Rozner, S. Banerjee, P. Bahl, "Dsap: A protocol for coordinated spectrum access," *DySPAN 2005*, pp. 611–614, 8–11 Nov. 2005.
24. L. Cao, H. Zheng, "Distributed rule-regulated spectrum sharing," *IEEE J. Selected Areas Commun.*, Vol. 26, No. 1, pp. 130–145, Jan. 2008.
25. X. Jing, D. Raychaudhuri, "Spectrum co-existence of IEEE 802.11b and 802.16a networks using reactive and proactive etiquette policies," *Mob. Netw. Appl.*, Vol. 11, No. 4, pp. 539–554, 2006.
26. R. Etkin, A. Parekh, D. Tse, "Spectrum sharing for unlicensed bands," *IEEE J. Selected Areas Commun.*, Vol. 25, No. 3, pp. 517–528, April 2007.
27. H. Harada, R. Prasad, "Simulation and software radio for mobile communications," *Artech House Universal Personal Communications Series*, 2002.
28. R. Jain, D. Chiu, W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," <http://arxiv.org/abs/cs.NI/9809099>, 1998.
29. IEEE Std 802.11e-2005, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements, 2005.

Index

A

Access segment, 157, 162
Action policy, 25
Architectural style, 32, 36, 37, 40, 41, 48, 52, 113
Architecture-based, 31–53, 262, 267
Artificial immune systems, 289–292
Attack prevention, 356, 377
Auctions, 99, 210, 211, 213–215, 216
Autonomic architecture, 5, 6–7, 9, 27, 110, 157, 267, 282
Autonomic computing (AC), 3–28, 34, 46, 91, 92, 101, 105–129, 131, 132, 134, 141, 151, 179, 181, 201, 202, 240, 241, 261, 285–286, 309, 335, 337, 341, 343, 350, 358, 364, 381, 382, 383, 384, 406
Autonomic-enabled ICT resource, 4
Autonomic-enabled web page, 25, 26
Autonomic management, 83–103, 131, 269, 270, 333–350
Autonomic networked computing (ANC), 381–407
Autonomic networking, 28, 157, 158, 159–162, 175, 176, 239–242, 244, 251, 253, 258, 261–282, 287–292, 310
Autonomic networks and systems, knowledge management, 239–258, 383, 384
Autonomic policy repository, 5
Autonomic system of systems, 24, 25
Autonomic Web Service Environment (AWSE), 335, 342–345, 346, 348, 349, 350
Autonomic wireless systems, 179–202
Autonomous mode, 169, 171, 172, 174
Availability, 9, 21–24, 25, 26, 33, 40, 52, 58, 69, 86, 110, 111, 129, 156, 157, 159, 161–162, 168, 169, 170, 172, 208, 226, 230, 246, 288, 289, 313, 314, 326, 328, 330, 339, 340, 341, 356, 436, 438

B

Behavioural model, 6, 9, 10, 12, 16, 17, 18, 20, 22, 23
Bio-Inspired machine learning, 131–152
Biologically-inspired networking, 140, 262, 287–292
Business dashboard, 25

C

Categorical approach, 388, 407
Churn attacks, 223, 224, 232, 233
Client-side management, 334, 345–348, 350
Closed-loop control, 33
Cognition and knowledge generation, 145, 148
Cognitive radio, 131–152, 179–184, 185, 186, 187, 188, 189–190, 193, 201, 436, 437
Component reuse, 4, 25
Components, 4, 6, 7, 8, 9, 10, 11, 16, 22, 25, 28, 34, 35 n.1, 36, 37, 39, 40, 42, 43, 44, 46, 48, 49, 83–103, 106, 107, 108, 109, 111, 112, 113, 115, 116, 126, 127, 128, 129, 181, 190, 191, 193, 194, 195, 208, 210, 215, 217, 229, 231, 240, 241, 242, 247, 249, 250, 251, 252, 254, 257, 258, 268–272, 275, 276, 278, 279, 280, 281, 286–287, 288, 292, 313, 314, 315, 316, 333, 335, 337, 339, 342, 343, 344, 347, 348, 349, 350, 382, 415, 416
Composite web services, 336
Comprehensive Service Management Middleware (CSMM), 335, 345–348, 349, 350
Connectors, 35 n.1, 36, 37, 40, 44, 99, 124
Continuous-time Markov chain, 12, 438
Convergence, 142, 144, 183, 250, 251, 252, 253, 254, 255, 256, 257, 421, 427–429, 432, 446
Core segment, 163

Cost–benefit attributes, 48–51, 53
CPU allocation, 12, 18, 19, 20, 102

D

Data centre, 21, 22, 23, 24, 25, 26
Data fusion, 57–78, 187, 188, 229
Delay Tolerant Networks (DTN), 411
Development methodology, 25, 27
Discovery service, 16 n.3, 114, 117, 118, 119, 120, 125
Distributed detection theory, 187, 188, 195, 196, 202
Distributed mode, 158, 166, 167, 169–172, 173–175
Distributed optimization, 156, 157, 165, 168, 421–422, 440, 443
Distributed Spectrum Sensing, 179–202
Distributed systems, 37, 105–108, 109–110, 112–113, 114, 116, 117, 119, 120, 126–129, 157, 175, 223, 241, 242, 250, 252, 334
Dynamic power management, 20, 23, 26
Dynamic spectrum access (DSA), 131–152, 436–440

E

Embodied cognition, 143, 146, 148, 179–202
Enterprise Application Integration (EAI), 336
Epidemic routing, 414, 418, 422
Euclidean metric, 16
Evolution, 32, 37, 38, 66 n.5, 84, 92, 142, 149, 161, 183, 190, 357, 363, 364, 403, 412, 415, 416, 417, 422, 423–425, 427, 428, 429, 430
Evolutionary computation, 412
Evolving protocols, 426
Exchange, 75, 115, 156, 170, 171, 172, 187, 207, 208, 211, 212, 215–217, 220, 224, 231, 233, 249, 273, 288, 324, 360, 365, 371, 374, 375, 376, 412, 413, 414, 417, 419, 423, 426, 432, 439, 446

F

Failure, 22, 36, 58, 83, 84, 86, 87, 91, 97, 100, 101, 106, 109, 110, 117, 140, 145, 157, 158, 162, 164–172, 181, 226, 228, 230, 240, 246, 247, 251, 263, 269, 274–281, 313, 345, 346, 348, 372, 374, 414
Fault Management, 265, 267, 273, 274–280, 334
Fault-tolerance, 316
Fitness, 299, 300, 364, 412, 413, 415, 416, 419, 420, 422–423, 424 n.2, 427, 431, 432

estimation, 412, 415
landscape, 431

Formal aspects, 381–407
Forwarding policy, 412, 415, 416, 417–419, 420, 424, 428, 432
Fujitsu disk drive, 20, 21

G

Game theory, 137, 242, 252–257, 258, 440
General-purpose autonomic computing, 3–28
Genetic algorithms, 64, 66–67, 141, 142, 184, 419, 421
Genotype, 412, 413, 416, 417, 418, 419, 420, 421, 422, 423, 424 n.2, 425, 428–432
GOAL operator, 22, 24
Goal policy, 18, 20

H

High-level effectors/sensors, 7
High-speed interconnection networks, network reconfiguration, 313–331

I

ICT(Information and Communication Technologies)
ontology, 27
resource, 4, 5, 6, 7, 8, 18, 20, 22, 25, 26, 27
IMS, 155–176
Incentives, 205–233, 371
Information diffusion, 411–432
Itinerary design, 69–70

J

J2EE, 10, 85, 86, 87, 89, 90, 92, 94, 95, 98, 99, 310

L

Legacy
ICT resource, 4, 6, 7, 18, 20
software, 83–103

M

Machine learning, 8, 9, 12, 52, 137, 142, 145, 146, 245, 248, 341, 347, 373
Manageability, 6, 7, 8, 10, 11, 12, 13–15, 23, 28, 76, 161, 344
Management Perspectives, 335, 342–349
MAPE, 34, 41, 337, 339, 340, 343, 345, 348, 358
Matlab, 26, 199
Medium access control (MAC), 64
Mobile agents, 57–78, 240, 270, 271–272, 273, 274
Mode identification, 189

Model-driven architecture, 6
 Model-driven autonomic computing, 6
 Model-driven development, 285–311
 Model driven engineering, 285–311
 Model-driven performance engineering,
 285–311
 Modifiability, 10, 13
 Multiple objectives, 39, 43, 47, 52
 Mutability, 10, 13
 Mutation, 142, 288, 296, 300, 301, 413, 416,
 421, 424, 425, 427, 429, 432

N
 Nash equilibrium, 220, 221, 222, 252–257
 .NET, 10
 Network
 management, 239, 240, 241, 243, 245, 246,
 247, 249, 257, 272, 273, 358
 protection, 371
 security, 356–357, 377

O
 OMNeT++, 427
 Online optimization, 132, 141, 142, 206, 223,
 246, 249, 333, 363, 415, 432
 Overload, 84, 86, 157, 158, 162, 168, 173, 174,
 175, 285, 286, 287, 338, 339, 359, 376

P
 Payments, 210, 211, 212–213, 215, 220, 231,
 232, 233
 Peer-to-Peer (P2P), 110, 156, 157, 158, 159,
 160, 205–233, 247, 248, 249, 251, 340,
 384
 Poisoning, 223, 224–226, 232, 233
 Policy-based autonomic computing, 4, 5
 Policy
 engine, 5, 6, 7, 8–9, 10, 11, 12, 15, 16, 18,
 20, 21–27
 sharing, 159, 205, 206, 207, 208, 217
 Pollution, 224–226, 233, 262
 PRISM, 10, 20, 21, 22
 Probabilistic model checker, 9, 10, 25

Q
 Quality dimensions, 32, 47, 48, 49, 50, 51
 Quality of service (QoS), 5, 52, 86, 91, 140,
 151, 156, 159, 161, 169, 207, 211, 219,
 252, 310, 334, 335, 338, 339, 340, 341,
 342, 345, 348, 350, 367, 444, 447
 Quantitative analysis, 10
 Quantitative model checking, 5

R
 Reciprocity, 211, 217–219, 233
 Reconfigurable policy engine, 5, 8–9
 Reputation systems, 211, 214, 232
 Resource-definition
 policy, 5, 11, 24, 25
 repository, 26, 28
 Reusability, 4, 28, 101
 Robustness, 140, 156, 157, 161–162, 169, 172,
 175, 200, 247, 248, 267, 313, 414, 415,
 430–431
 Routing, 60 n.2, 67, 73, 74, 75, 136, 159, 163,
 165, 168, 170, 171, 243, 253, 256, 265,
 266–267, 270, 271, 272, 273, 275, 276,
 279, 280, 281, 314, 315, 316, 318, 319,
 320, 321, 322, 323, 326, 327, 328, 329,
 330, 331, 358, 367, 371, 375, 412, 413,
 414, 418, 422
 protocols, 265, 266, 414
 Runtime code generator, 8

S
 SBC.PROXY, 170, 171, 172
 Scalability, 59, 61, 62, 68, 73, 85, 208, 230,
 246, 247, 249, 263, 340, 430, 440, 443,
 444, 448
 Selection process, 52, 289, 297, 419, 422
 Self-*, 4, 26, 27, 157, 241, 381–407
 Self-adaption, 37
 Self awareness, 106, 108, 110, 111, 112–113,
 116, 120, 126–127, 128, 132, 134, 184,
 240, 241, 250, 261, 383
 Self discovery, 106, 108, 110, 111, 112–113,
 117, 119, 120, 126–127, 128, 129
 Self-managing system, 4, 5, 6, 7, 9, 11, 18, 25
 Self-optimisation, 4, 20, 26
 Self-stabilization, 239–258
 Server allocation, 23, 338
 Server-side management, 334, 342–345,
 348, 349
 Service level agreements (SLA), 168, 176,
 286, 334, 338, 342, 343, 344, 345, 346,
 347, 348, 349, 350
 Service oriented architecture (SOA), 106, 107,
 113–115, 116–117, 119–120, 333
 Session Border Controller (SBC), 162, 163,
 164, 165, 166, 167, 169–175
 SIP (Session Initiation Protocol), 163, 173
 Stateful web services, 117–118, 121, 123, 125
 Stateless web services, 125–126
 Strategy, 34, 39, 41, 42, 43, 46, 47, 48, 49, 51,
 67, 133, 137, 138, 140, 142, 144, 150,
 157, 180, 185, 214, 228, 249, 253, 254,

255, 268, 314, 315, 316, 326, 330, 339, 435, 441, 447

Subscribeability, 10, 11, 13

Sybil attacks, 223, 226–228, 232, 233

System

- configuration, 11, 40, 240, 339
- meta-model, 11, 13, 27
- model, 7, 8, 9, 11, 12 n.2, 13, 15, 16, 18, 20, 21, 22, 28, 41
- state, 8, 38, 45, 51, 111, 148, 250, 251, 252, 253
- of systems, 24, 25, 27

T

Trade-off, 22, 23, 35, 51, 78, 131, 140, 281, 367, 412, 414, 418, 420, 428

U

UDDI (Universal Description Discovery and Integration), 107, 115, 117, 118, 119, 336, 347, 349

Universal policy engine, 6

Utility

- function, 4, 5, 9, 10, 11, 12, 16, 17, 18, 19, 20, 21, 22 n.5, 23, 26, 47, 50, 215, 231, 253, 443
- policy, 4, 5, 9, 10, 11–12, 16, 18, 19, 20
- preferences, 48, 50, 51, 53

V

VoIP, 156, 157, 163, 209

W

Web Service Resource Framework (WSRF), 107, 118, 120, 121, 123, 125–126, 338, 339, 341

Web service/services, 6, 10, 11, 15, 107, 112, 113, 114, 115–118, 119, 120–128, 305, 333–350

- management, 334, 335, 344, 348

Whitewashing, 211, 218, 219

Wireless network coexistence, 141, 207, 232, 256, 261–262, 282, 365, 384, 411, 436, 440, 443, 444, 446, 447, 448

Wireless networks, 141, 207, 232, 256, 262, 282, 365, 384, 436, 440, 443, 444, 446, 447, 448

Wireless sensor networks, 57–78, 142, 261–282

WSDL document, 115, 118, 119, 120, 121, 336

X

XML Schema, 9, 13, 14

XML Schema Definition tool, 14

XSLT transformation, 13