Service-Oriented Software System Engineering

Challenges and Practices

Zoran Stojanovic & Ajantha Dahanayake

Service-Oriented Software System Engineering:
Challenges and Practices

Zoran Stojanovic
Delft University of Technology, The Netherlands

Ajantha Dahanayake
Delft University of Technology, The Netherlands

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore

Acquisitions Editor: Mehdi Khosrow-Pour
Senior Managing Editor: Jan Travers
Managing Editor: Amanda Appicello
Development Editor: Michele Rossi
Copy Editor: April Schmidt
Typesetter: Jennifer Wetzel
Cover Design: Lisa Tosheff
Printed at: Integrated Book Technology

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Service-oriented software system engineering : challenges and practices / Zoran Stojanovic and Ajantha Dahanayake, editors.
p. cm.
Includes bibliographical references and index.
ISBN 1-59140-426-6 (h/c) -- ISBN 1-59140-427-4 (s/c) -- ISBN 1-59140-428-2 (ebook)
1. Software engineering. I. Stojanovic, Zoran, 1969- II. Dahanayake, Ajantha, 1954-
QA76.758.S458 2004
005.1--dc22
2004021990

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Service-Oriented Software System Engineering:
Challenges and Practices

Table of Contents

Preface

Components and Web Services

Modern enterprises are caught in the flux of rapid and often unpredictable changes in business and information technology (IT). New business demands caused by an enterprise's need to be competitive in the market require the immediate support of advanced IT solutions. At the same time, new IT opportunities and achievements are constantly emerging and must be rapidly adopted to support new and more effective ways of conducting business. Therefore, it is crucial to provide effective business/IT alignment in terms of producing high quality and flexible IT solutions within a short time-to-market that exactly match business functionality needs and change as business changes. During the last few years, there has been a growing consensus in the industry that the way to create these adaptive, business-driven IT solutions is to use discrete building blocks of software, which are based on industry-standard protocols and interoperate across platforms and programming languages.

Component-based development (CBD) (Brown & Wallnau, 1998) and then Web services (Barry, 2003) have been proposed as ways to build complex but adaptive and agile enterprise information systems that provide effective inter- and intra-enterprise integration. The CBD platforms and technologies, such as CORBA Components, Sun's Enterprise Java Beans (EJB), and Microsoft's COM+/.NET, are now *de facto* standards in complex Web-based systems development. Further, a growing interest in Web services has resulted in a number of industry initiatives to provide platform-independent communication of software resources across the Internet (W3C, 2004). Basic elements of the new Web services paradigm are the standards for interoperability — XML, SOAP, WSDL and UDDI (Newcomer, 2002). On top of this basic interoperability protocol stack, new languages and specifications for defining the composition of services to form real-world business processes have emerged, such as Business Process Execution Language for Web Services (BPEL4WS) (BPEL, 2003) and Web Service Choreography Interface (WSCI) (Arkin et al., 2002).

Using this advanced technology, the Internet, once solely a repository of various kinds of information, is now evolving into a provider of a variety of business services and applications. Using Web services technology, organizations are now provided with a way to expose their core business processes on the Internet as a collection of services.

Customers and business partners are potentially able to invoke and retrieve these services over the Internet and compose them as wished to achieve their business goals. This idea of a software application as a service was recognized in the past (as in Brown, 2000), but it can now be fully realized using the Web services technology for systems interoperability (Newcomer, 2002). Web services can be considered the technological foundation for the service-oriented computing paradigm. The W3C's Web Services Architecture Working Group in its Web Services Glossary (2004) defines a Web service as:

> *a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Is Technology Enough?

A common point for both CBD and Web services paradigms is that they are technology led, that is, they were originally introduced through the new technology standards, infrastructures, and tools. Although technology is essential in building complex IT solutions from components and services, it is not sufficient on its own to support the full extent of an enterprise's business and IT requirements. Application functionality is routinely "packaged" into components today; however, the essential design and development methods and processes that enable application adaptability, widespread reuse, and commercialization still have little acceptance (Stojanovic, Dahanayake & Sol, 2004).

As using component middleware technology does not ensure that one will achieve the promised benefits of the CBD approach, conversely, the CBD paradigm can be successfully employed without using component middleware technology. A similar issue now arises in the case of Web services. The standards and protocols for Web services are well established and increasingly used. However, developing a coherent set of design and process principles for engineering service-oriented solutions throughout the development life cycle, which is crucial for achieving the full benefits of service orientation, is still at an immature stage (Kaye, 2003). As there is more to CBD than packaging software into Java Beans or .NET components, there is more to service orientation than simply rendering interfaces of software entities as Web services.

Another critical issue in today's enterprise IT developments is that the available set of technologies for components, Web services and business process automation, orchestration and integration is complex and constantly evolving. This can cause problems whenever new versions of technology standards and interoperability protocols appear. Moreover, developing systems directly using these technologies is tedious, complex, and error prone.

Therefore, the real challenge lies not just in new technology but also in how best to make use of the available technology through systems engineering principles and practices, from service identification and specification to service deployment. Applying well-defined design and engineering methods and techniques ensures that we do not end up with a random collection of unusable, although technologically feasible, services. Equally important is a conceptual service model that provides a precise definition of the underlying concepts used in service-oriented computing including service, component, interface, collaboration, port, and so forth.

Further, we need to develop systems to a higher level of abstraction that make a development process more productive, flexible, and understandable for business people who define requirements, use the solutions, and decide about future strategies. There is a strong need for service-oriented modeling, design and development methods, and techniques that will map high-level business requirements to software technology implementation and bridge the gap between business and IT (Apperly et al., 2003; Atkinson et al., 2002; Herzum & Sims, 2000). To make components and Web services a prominent and mainstream paradigm in building enterprise-scale information systems, well-defined engineering principles and techniques are required to maximize the benefits of service orientation and plug-and-play mechanisms.

The idea of software systems as the collaboration and coordination of components that provide services represents an interesting perspective with a number of new software engineering challenges. Service-oriented software engineering (SOSE) is concerned with theories, principles, methods, and tools for building enterprise-scale solutions as the collaboration of loosely-coupled application services that provide particular business functionality and are distributed within and across organizational boundaries. Important topics that need to be addressed within the SOSE paradigm include but are not limited to:

- precise definitions of service-related concepts that are applicable throughout the development life cycle, specified at the level of analysis and design, and successfully refined into implementation and deployment artifacts.
- standard modeling and specification notations for representing service concepts in graphical, human-understandable, and/or machine-readable formats.
- development methods and processes based on service-oriented and component-based ways of thinking, organized around the concepts of service and component.
- the way of designing the service-oriented enterprise system architecture from various perspectives and viewpoints that reflect the needs of various stakeholders.
- deployment of the service-oriented system architecture onto available technology infrastructure.

Engineering service-oriented solutions need to address the concerns of both the service provider and the service consumer. From the service provider perspective, it is important to define what component of the system can be exposed as a service, offering a business value to the consumer, and at the same time is, as much as possible, decoupled

from the rest of the system. From the service consumer perspective, it is important to determine what part of the system logical architecture can be realized by invoking a particular service over the Web and how that part can interface with the existing organization's system services and components. Balancing the needs of service provider and consumer is crucial to achieving the true benefits of service orientation for business agility and inter- and intra-enterprise integration.

Principles of Service Orientation

The central point of the SOSE paradigm is the concept of *service*, as well as the strategy to expose system capabilities to consumers as services through the Service-Oriented Architecture (SOA). In this respect, the Web services technology is just a way to efficiently realize the concepts of services and SOA. The service forms a contractual agreement between provider and consumer. Besides a common interface that defines operation signatures, a service can also have attributes of its own, such as service level agreement, policies, dependencies, and so forth.

A service interface defines a contract and the parties' obligations precisely and, thus, allows the consumer to use the functionality offered without being aware of the underlying implementation. As defined by the W3C's Web Services Glossary (2004), a service is "an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities. To be used, a service must be realized by a concrete provider agent." There are actually a number of parallels between service orientation and classical CBD. Like components, services represent natural building blocks that allow us to organize the capabilities of a system in ways that are meaningful to the user of the system. Similar to components, a service combines information and behavior, hides the internal workings from the outside perspective, and presents a relatively simple interface to the environment (Kaye, 2003). When using Web services technology, the component itself is not acquired in the traditional manner of taking a copy and executing it in the house but rather just the services provided by the component are consumed via the Web while the component executes its function at a single location, available to all who subscribe (Newcomer, 2002).

In its essence, SOA is a way of designing a software system to provide services to either end-user applications or to other services through published and discoverable interfaces (Kaye, 2003). As defined by the W3C's Web Services Glossary (2004), SOA is "a set of components which can be invoked and whose interfaces descriptions can be published and discovered." A basis of SOA is the concept of service as a functional representation of a real-world business activity that is meaningful to the end user and encapsulated in a software solution. Using the analogy between the concept of service and business process, SOA provides for loosely coupled services to be orchestrated into business processes that support business goals. Initiatives similar to SOA were proposed in the past, such as CORBA or Microsoft's DCOM. What is new about SOA is that it relies on universally accepted standards like XML and SOAP to provide broad interoperability among different vendors' solutions, and it is based on the proven component-based design principles and techniques.

The power and flexibility that SOA potentially offer to an enterprise are substantial. If an enterprise abstracts its IT infrastructure and functionality in the form of coarse-grained services that offer clear business value, then the consumers of those services can access them independent of their underlying technology and use them to achieve business goals. In essence, services and SOA act as a layer of abstraction between the business and the technology. The dynamic relationships between the needs of the business and the available services, on the one hand, and the technology foundation that realizes and supports the services, on the other hand, must be well understood and designed. Therefore, one of the main tasks of new service-oriented software engineering concepts and principles is to help achieve effective business-IT alignment based on the concept of service as a common ground.

Related Topics

In today's world of continually changing business and IT, it is crucial to decide what strategy and process to follow in engineering complex service-oriented software systems. In the last few years, two increasingly important movements in IT, corresponding to fundamentally different philosophies about how software systems should be built, have emerged. The first, model-driven development, tries to preserve investments in building systems against constantly changing technology solutions by creating formal architecture models that can be mapped to whatever software technology. The Object Management Group (OMG) has proposed Model Driven Architecture (MDA) as an attempt to raise the level of abstraction in software development as a well-established trend in computing (Frankel, 2003). MDA separates the concerns of the business specification from the details of the technology implementation. System development using the MDA approach is organized around a set of models by imposing a series of transformations between the models (OMG-MDA, 2003). A formal foundation for describing the models is the set of OMG standards — UML, MOF, XMI, CWM (specifications available at http://www.omg.org) — that facilitate meaningful integration and transformation among the models, and are the basis for automation through tools. New developments that support the MDA paradigm are the standard UML profile for Enterprise Distributed Object Computing (EDOC) and the new major revision of the UML, version 2.0 (OMG-UML, 2004). These standard specifications propose new concepts and ideas regarding the way components are defined at the logical level, making a solid foundation for modeling and design of services and service-oriented applications.

Parallel to the MDA initiative, the last few years have witnessed considerable interest in the IT community for eXtreme Programming (XP) and other methodologies for Agile Development (AD) (Cockburn, 2002). Agile processes are focused on early, fast, and frequent production of working code using fast iterations and small increments. The processes are characterized by intensive communication between participants, rapid feedback, simple design, and frequent testing. Proponents of AD see the software code as the main deliverable, while the roles of system analysis, design and documentation in software development, and maintenance are de-emphasized and, to some extent, ignored.

While both AD and MDA claim to address the challenges of high change rates, short time-to-market, increased return-on-investment, and high quality software, it is obvious that their proposed solutions are actually very dissimilar. MDA assumes mainly fixed user requirements and suggests creating formal models around them, whereas AD handles constantly changing user requirements using fast and frequent prototyping. It is challenging to determine in what ways the principles and practices of both development paradigms can be combined in engineering service-oriented systems to gain the potential benefits of both approaches.

Another interesting development related to the SOSE paradigm is the Business Process Management Initiative (BPMI) (http://www.bpmi.org) the main purpose of which is to define ways for enabling computer-assisted management of business processes. The BPMI has issued the specification of Business Process Modeling Language (BPML), Business Process Modeling Notation (BPMN), and Business Process Query Language (BPQL) that provide for the management of different aspects of e-business processes that span multiple applications, corporate departments, and business partners over the Internet. BPML has a lot in common with and builds on the foundations of the Web services composition languages such as BPEL and WSCI. Moreover, BPMN is very much related to the new diagram set of UML 2.0 that represents action semantics. Hence, some joint efforts towards unified standard specifications and notations can be expected in the near future.

The new developments in the field of Web services have provided a lot of possibilities but have also introduced new challenges. Novel mobile technologies provide new channels for providing and consuming services in a wireless setting, anytime and anywhere. Furthermore, for Web services to become a reality across enterprises, security and trust issues in providing and consuming services across the Web must be settled at a much higher level. The earlier dilemma of a software developer regarding reusing software and how far to trust somebody else's software code is now largely substituted by the dilemma of a business analyst regarding how far to trust somebody else's services.

Summary

The purpose of this book is to survey the main concepts, principles, practices, and challenges of the new paradigm of service-oriented software engineering (SOSE). The series of papers included in this book show the wide variety of perspectives on SOSE and illustrate the wide-ranging impact that SOSE can have on how complex enterprise information systems are built today and will be built in the future, when attempting to hit the moving target of continuously changing business needs.

As illustrated throughout this book, the new SOSE paradigm can provide a meeting point between business process management and automation on the one side and component-based software engineering on the other, bridging the gap between business and IT. The aim of this book is to disseminate the research results and best practices from researchers and practitioners interested in and working on different aspects of SOSE, setting up a new agenda for the exciting world of services.

One of the strengths of this book is its international flavor. The authors of the various chapters come from various countries worldwide. This gives the reader a range of perspectives on the issues taken from different world viewpoints. Although a number of books about Web services have already been published, this book is one of the first that goes beyond the pure technology level of the Web services protocols. The book presents innovative and effective service-oriented software engineering concepts, principles, techniques, and practices that can be followed throughout the development life cycle in order to fulfill the great promises of service orientation. We believe that this book can serve as a starting point for new, interesting, and challenging developments in the exciting area of SOSE.

Organization of the Book

The book consists of 18 chapters, organized into five sections. A brief description of each of the chapters follows.

The first section of the book presents core service-oriented concepts and technologies as a basis for the rest of the book.

Cervantes and Hall (Chapter I) present service-oriented concepts from a technological perspective to position them with respect to those present in component orientation and to illustrate how they are realized. The technical presentation is followed by a survey of several service-oriented platform technologies including CORBA Traders, JavaBeans Context, Jini, OSGi, and Web services.

Zhou and Niemelä (Chapter II) introduce service-oriented software engineering as an advanced software development. The authors present SOSE software development methodology involving the main processes of service extracting, middard, circulation, evaluation, and evolution with the middard service fundamental.

Benatallah et al. (Chapter III) provide an overview of the area of service composition. The chapter presents a critical view into a number of languages, standardization efforts, and tools related to service composition and classifies them in terms of the concepts and techniques that they incorporate or support. It discusses some trends in service-oriented software systems engineering pertaining to service composition.

The second section of the book deals with different aspects of service-oriented modeling, architecture, design, and development.

Olson et al. (Chapter IV) introduce the UniFrame approach for creating high quality computing systems from heterogeneous components distributed over a network. Their chapter describes how this approach employs a unifying framework for specifying such systems to unite the concepts of service-oriented architectures, a component-based software engineering methodology, and a mechanism for automatically finding components on a network to assemble a specified system.

Latchem and Piper (Chapter V) present a worked example of a design process for service-oriented architecture. The process utilizes the industry standard modeling notation, the Unified Modeling Language (UML) from the Object Management Group, to present a practical design for services.

Piccinelli and Skene (Chapter VI) introduce the model-driven architecture (MDA) concept and technologies to the service-oriented computing (SOC) paradigm and employs these technologies to enhance support for SOC through the definition of a domain-specific modeling language for electronic services. The language is defined as an extension of the Unified Modeling Language (UML).

Steen et al. (Chapter VII) study the relevance and impact of the service concept and service orientation to the discipline of enterprise architecture. This chapter argues that a service-oriented approach to enterprise architecture provides better handles for architectural alignment and business and IT alignment, in particular.

Kotonya et al. (Chapter VIII) present a negotiation-driven method that can be used to formulate and design component- and service-oriented systems. The software engineering method is capable of balancing aspects of requirements with business concerns and the architectural assumptions and capabilities embodied in software components and services.

Král and Žemlička (Chapter IX) discuss the crucial elements of the requirements specification of service-oriented software systems as well as the relation between the requirements specification and the architecture of these systems. The chapter shows that there are several variants of service-oriented software systems having different application domains, user properties, development processes, and software engineering properties.

Constantinides and Roussos (Chapter X) introduce service patterns for service-oriented enterprise systems. The authors argue that the deployment of such patterns would be of considerable value as a best-practice guide for practitioners and a starting point for further research in their role in software engineering. A comprehensive catalog of service patterns is included in this chapter as well as recommendations on their implementation and a practical usage scenario.

The third section of the book is concerned with service-oriented computing in the wireless and mobile settings and agent-based services.

Maamar (Chapter XI) argues that enacting Web services from mobile devices and possibly downloading these Web services for execution on mobile devices are avenues that academia and industry communities are pursuing. The author presents two research initiatives carried out at Zayed University. SAMOS stands for Software Agents for MObile Services, and SASC stands for Software Agents for Service Composition.

Sen et al. (Chapter XII) introduce an ad hoc wireless network as a dynamic environment, which exhibits transient interactions, decoupled computing, physical mobility of hosts, and logical mobility of code. The authors examine the imperatives for a viable service-oriented computing framework in ad hoc wireless settings.

Li et al. (Chapter XIII) propose service-oriented agents (SOAs) to unify Web services and software agents. Web services features can be well realized through introducing sophisticated software modeling and interaction behaviors of software agents. A prototype of the proposed SOAs framework has been implemented.

The fourth section of the book deals with an important topic of security in engineering service-oriented systems, which is an essential prerequisite for wide use of services across the Web.

Padmanabhuni and Adarkar (Chapter XIV) examine the security requirements in SOA implementations and discuss the different solution mechanisms to address these requirements. The chapter critically examines the crucial Web services security standards in different stages of adoption and standardization as well as today's common nonstandard security mechanisms of SOA implementations.

Phillips et al. (Chapter XV) examine the attainment of advanced security capabilities using the middleware paradigm, namely, role-based access control (RBAC) and mandatory access control (MAC). The resulting security provides a robust collection of services that is versatile and flexible and easily integrates into a distributed application comprised of interacting legacy, COTS, GOTS, databases, servers, clients, and so forth.

The final, fifth section of the book presents service-oriented solutions in several application domains that show the whole strength of the new service-oriented computing paradigm in building today's complex Web-based software systems.

Janssen (Chapter XVI) presents the design of a service-oriented architecture in public administration. A case study is conducted at the Ministry of Justice, and a service-oriented architecture is designed, implemented, and evaluated based on a number of quality requirements. This case study shows the feasibility replacing functionality formerly offered by legacy systems, limitations of current technology, and promises of applying service orientation successfully in complex domains, such as e-government.

Dustdar et al. (Chapter XVII) present a sound and flexible architecture for gluing together various Groupware systems using Web services technologies. The chapter presents a framework consisting of three levels of Web services for Groupware support, a novel Web services management and configuration architecture for integrating various Groupware systems, and a preliminary proof-of-concept implementation.

Wu and Subramanium (Chapter XVIII) present a case study where Web services are used to build a user-centric online security system. It explores complicated technical challenges encountered with the use of the Web services and online security technology. The authors argue that their practical experiences and findings can provide more insight on how the online security system can be built in a user-centric, instead of vendor-centric, way by using Web services on top of conventional software engineering processes.

References

Apperly, H. et al. (2003). *Service- and component-based development: Using the select perspective and UML.* Boston: Addison-Wesley.

Arkin, A. et al. (Eds.). (2002). Web Service Choreography Interface (WSCI) 1.0. Retrieved August 2, 2004: *http://www.w3.org/TR/wsci/*

Atkinson, C. et al. (2002). *Component-based product line engineering with UML*. Boston: Addison-Wesley.

Barry, D. K. (2003). *Web services and service-oriented architectures: The savvy manager's guide.* San Francisco: Morgan Kaufmann.

BPEL. (2003). Business process execution language for Web services version 1.1. Retrieved August 2, 2004: *http://www-106.ibm.com/developerworks/library/ws-bpel*

Brown, A.W. (2000). *Large-scale component-based development*. Indianapolis, IN: Prentice Hall PTR.

Brown, A.W., & Wallnau, K.C. (1998). The current state of CBSE. *IEEE Software, 15*(5), 37-46.

Cockburn, A. (2002). *Agile software development.* Boston: Addison-Wesley.

Frankel, D. S. (2003). *Model driven architecture: Applying MDA to enterprise computing*. Indianapolis, IN: Wiley.

Herzum, P., & Sims, O. (2000). *Business component factory: A comprehensive overview of component-based development for the enterprise*. Indianapolis, IN: Wiley.

Kaye, D. (2003). *Loosely coupled: The missing pieces of Web services*. Kentfield, CA: RDS Press.

Newcomer, E. (2002). *Understanding Web services: XML, WSDL, SOAP and UDDI.* Boston: Addison-Wesley.

OMG-MDA (2003). Model driven architecture. Retrieved August 2, 2004: *http://www.omg.org/mda/*

OMG-UML (2004). UML™ resource page. Retrieved August 2, 2004: *http://www.uml.org/*

Stojanovic, Z., Dahanayake, A., & Sol, H. (2004). An evaluation framework for component-based and service-oriented system development methodologies. In K. Siau (Ed.), *Advanced topics in database research, volume 3* (pp. 45-69). Hershey, PA: Idea Group.

W3C (2004). W3C World Wide Web consortium. XML, SOAP, WSDL specifications. Retrieved August 2, 2004: *http://www.w3c.org/*

W3C Web Services Glossary. (2004). W3C group note. Retrieved August 2, 2004: *http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/*

Acknowledgments

We would like to acknowledge the help of all involved in the collation and review process of the book without whose support the project could not have been satisfactorily completed. Obviously, in any project of this size, it is impossible to remember, let alone mention, everyone who had a hand in this work becoming what it is today.

First, we wish to thank all of the authors. They deserve the greatest credit because their contributions were essential, giving us great material with which to work. It was a wonderful experience to work with them, to read their contributions, and to discuss the book's overall objectives and particular ideas. Most of the authors of chapters included in this book also served as referees for articles written by other authors. Thanks go to all those who assisted us in the reviewing process by providing constructive and comprehensive reviews.

Staff members of Systems Engineering and Information & Communication Technology groups at the Faculty of Technology, Policy and Management at Delft University of Technology were critical in creating this final product. Their support was vital in achieving what we hope is a well-edited publication.

A special note of thanks goes to all the staff at Idea Group Inc., whose contributions throughout the whole process, from inception of the initial idea to final publication, have been invaluable. In particular, we thank Michele Rossi who continuously prodded via e-mail to keep the project on schedule and Mehdi Khosrow-Pour whose enthusiasm motivated us to accept the invitation to take on this project.

Finally, we wish to express our gratitude to our families for their unfailing patience, support, and love. Our thanks to all these people!

Zoran Stojanovic & Ajantha Dahanayake
Delft, The Netherlands
2004

Section I

Core Service Concepts and Technologies

Chapter I

Technical Concepts of Service Orientation

Humberto Cervantes
Laboratoire LSR Imag, France

Richard S. Hall
Laboratoire LSR Imag, France

Abstract

This chapter presents service-oriented concepts from a technological perspective. Before delving into service orientation, concepts in component orientation are introduced for a point of reference. After, service orientation is introduced via the service-oriented interaction pattern and the entities that participate in it, followed by a discussion of how these entities and service orientation, in general, relate to component orientation. The technical presentation is followed by a survey of several service-oriented platform technologies, including: CORBA Traders, JavaBeans Context, Jini, OSGi, and Web services. The purpose of this chapter is to present service-oriented concepts from a technological perspective, position them with respect to those present in component orientation, and illustrate how they are realized.

Introduction

Service orientation is a trend in software engineering that promotes the construction of applications based on entities called *services*. The notion of a service is, however, not concretely defined and can represent different concepts to different stakeholders. From a coarse-grained point of view, services are activities that are realized by an application,

machine, or human being. While this point of view is helpful (for example when modeling an enterprise), it is somewhat distant from application development concepts, where a service is a *reusable* building block that offers a particular functionality. In this context, the term *reusable* means that the same service can be used to construct multiple applications. The notion of reusability evokes similarities to *component orientation*, which is another software development approach that also promotes the idea of constructing applications from the assembly of reusable building blocks called *components* (Meijler & Nierstrasz, 1997). Both service and component orientation share a common development model where building block development and assembly are performed by different actors and can take place at different locations (that is, different enterprises). Service orientation focuses on other aspects, though, such as the support for dynamic discovery, that are generally not explicit considerations of component orientation.

This chapter presents service orientation from a more technical, fine-grained point of view. It starts by briefly presenting concepts associated with component orientation. Following this, the concepts of service orientation are discussed and then compared to those of component orientation. Finally, a set of technologies that support the service-oriented approach are surveyed. The surveyed technologies include CORBA Traders (Stratton, 1998), JavaBeans Context (Sun Microsystems, 1998), Jini (Arnold, O'Sullivan, Scheifler, Waldo & Wolrath, 1999), Open Services Gateway Initiative framework (OSGi) (Open Services Gateway Initiative, 2003), and Web services (Curbera, Nagy & Weerawarana, 2001). The chapter concludes with a discussion about the ideas contained herein.

Component Orientation

This section presents the concepts of component orientation based on concepts present in a series of component technologies, which include JavaBeans (Sun Microsystems, 1997), Microsoft's Component Object Model (COM) (Box, 1998), Enterprise Java Beans (EJB) (Sun Microsystems, 2001), and the CORBA Component Model (CCM) (Object Management Group, 2003).

Terminology

Although there is no universal agreement on a definition for the term *component*, the definition formulated by Szyperski (1998) is widely referenced in literature:

> *A software component is a binary unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

This definition contains several important concepts that characterize a component. In the context of this chapter, however, this definition is refined by specifying that a component can be instantiated to produce *component instances* and is independently delivered and deployed in a *component package*. A component model defines a set of characteristics regarding components, compositions, and their supporting *execution environment*.

Component Elements

Component characteristics are realized by three different elements: components, component instances, and component packages.

- **Component:** A component is similar to a class concept in object orientation in the sense that instances can be created from it. To support composition, a component exposes an *external view* that contains a set of *functional interfaces* that are provided or required along with a set of *configuration properties*. Interfaces are categorized as being functional since they only contain methods that are related to the component's functionality. The external view is implemented by a component implementation that can expose a set of additional *control interfaces* and *deployment dependencies*. Control interfaces enable the execution environment to manage a component instance's life cycle (this is discussed later), while deployment dependencies represent, for example, dependencies toward a particular version of the execution environment or a needed resource.
- **Component instance:** A component instance is obtained from a component; in object-oriented terms, it is equivalent to an object since it has a unique identifier and may have modifiable state. A component instance is configured and connected to other component instances inside a *composition*.
- **Component package:** A component package is a unit that allows components to be delivered and deployed independently. The term independent refers to the fact that the component package contains everything that is needed by the component to function (for example, resources such as images, libraries, configuration files) with the exception of anything that is declared as an explicit dependency (for example, required functional interfaces).

Composition

In component orientation, applications are assembled from components and assembly is achieved through component composition. A composition description is used during execution to create, configure, and connect a set of component instances that form an application. The fact that architectural information is located in the composition description and not inside the component's code facilitates component reuse. Compositions can be created in different ways: visually, in a dedicated environment such as the BeanBuilder from Sun (Davidson, 2002), declaratively, through a language such as an Architecture

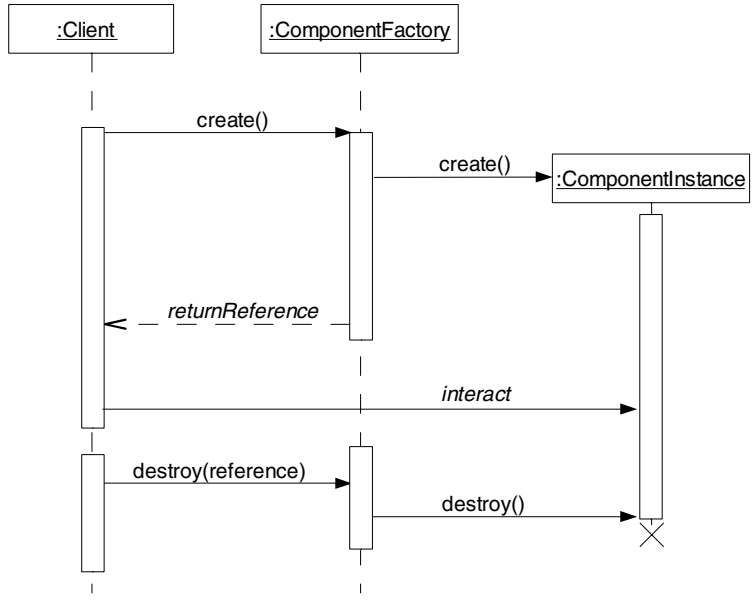
Description Language (Clements, 1996), or imperatively, through languages such as system or scripting languages (Ousterhout, 1998).

Execution

During execution, component instances are typically created and destroyed through factories following an interaction pattern depicted in Figure 1. Factories decouple clients from particular component implementations. Additionally, factories allow for different instance creation policies. For example, an instance can be a singleton that is shared among all clients or instances can be allocated from an instance pool on demand.

When a component instance is created, its life cycle is usually managed by a *container* (Conan, Putrycz, Farcet & DeMiguel, 2001), which is an entity from the execution environment that wraps a component instance. The container manages an instance by invoking methods defined in the control interfaces following the inversion of control pattern (Fowler, 2004). These control methods allow, for example, instance execution to be suspended, instance state to be persisted, or instance to be reconfigured. An application may also impose a set of *nonfunctional* requirements on its constituent components; examples of such requirements include security, performance, or distribution. These requirements can be handled by the container on behalf of the components by intercepting the calls made to the component instance.

Figure 1. Instance creation interaction pattern



Service Orientation

Although the popularity of service orientation has increased with the emergence of Web services, service-oriented principles were present in the *trading* mechanisms of distributed systems, such as in the ODP trader (Indulska, Bearman & Raymond, 1993).

The following subsections present service-oriented concepts by defining the word *service*, describing the service-oriented interaction pattern, presenting the elements that constitute a service, and introducing the necessary execution environment to support the service-oriented approach. The section concludes with a comparison to component orientation.

Terminology

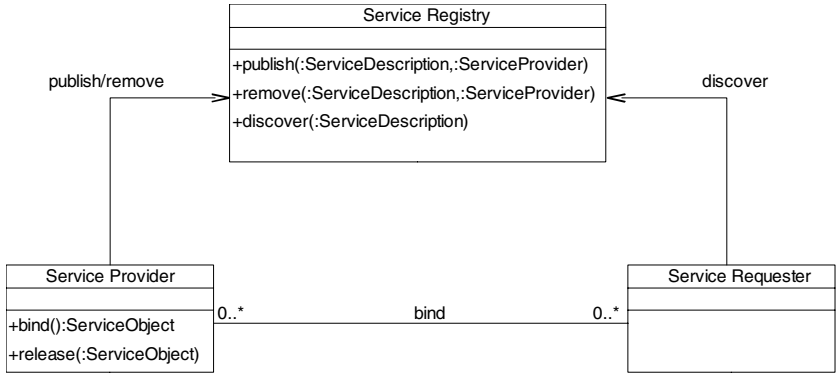
A *service* offers reusable functionality that is contractually defined in a *service description*, which contains some combination of syntactic, semantic, and behavioral information. In service orientation, application assembly is based only on service descriptions; the actual *service providers* are discovered and integrated into the application later, usually prior to or during application execution. As a result, service orientation focuses on how services are described in a way that supports the dynamic discovery of appropriate services at run time (Burbeck, 2000). Service orientation promotes the idea that a service requester is not tied to a particular provider; instead, service providers are substitutable as long as they comply with the contract imposed by the service description. An important assumption in service orientation is that services may be *dynamically available*, that is, their availability can vary continuously.

Service-Oriented Interaction Pattern

To support dynamic discovery, service orientation is based on an interaction pattern that involves three different actors, depicted in Figure 2:

- **Service provider:** The service provider is responsible for supplying service objects that implement service functionality.
- **Service requester:** The service requester is a client for a particular service.
- **Service registry:** The service registry is an intermediary between service requesters and service providers. The registry contains a set of service descriptions along with references to service providers; it provides mechanisms for service publication, removal, and discovery. The set of service descriptions contained in the service registry changes as services provided by service providers are published and removed from the registry.

Figure 2. Actors of the service-oriented interaction pattern

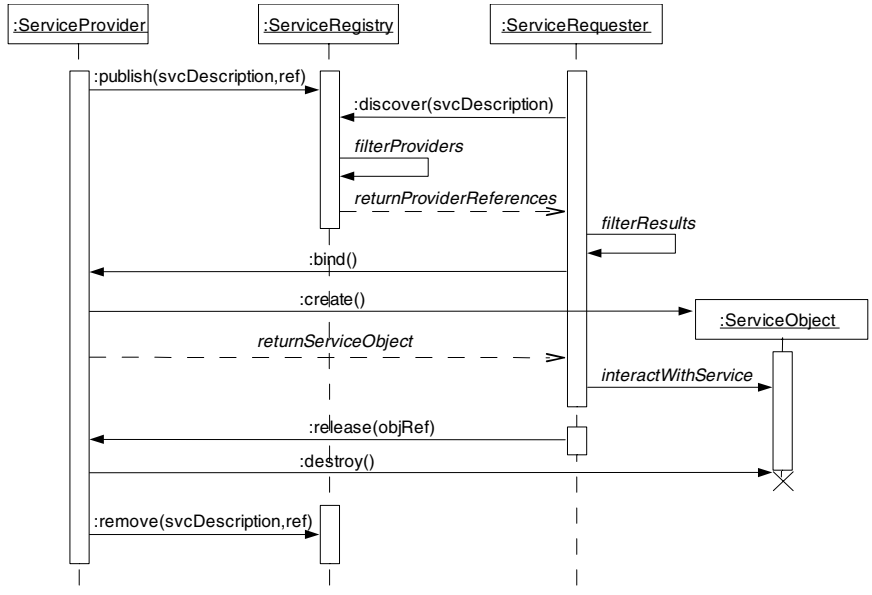


The basic interaction pattern that characterizes service orientation is depicted in the sequence diagram in Figure 3. This diagram shows a service provider that *publishes* a service description in a service registry. A service requester further queries the service registry to *discover* services based on a set of criteria relative to a service description. If service providers that meet the criteria have been previously published, the service registry returns the provider's references to the service requester. In the case where multiple answers are returned, the service requester may need to select a specific service provider to which it will *bind*. When the service requester binds to the provider, a service object is returned by the provider. Finally, when the service requester finishes interacting with the service, the service object is either implicitly or explicitly released.

Service Description

As previously mentioned, the service description combines syntactic, semantic, and behavioral information. The syntactic part of a service description is typically embodied as a *service interface*, which is a set of operations that provide the functionality of the service. A service interface defines a syntactic contract and also provides a limited level of semantics from its operation names; for example, a method named print() in a printer service interface. It is common that service-oriented technologies rely solely on syntactic descriptions; this requires, however, that consensus or standards organizations define the exact behavior of a service interface, which is then described in separate specification documents that are intended for humans. This approach is potentially impractical, since building consensus on every service interface is not always possible or desirable. Much research exists in explicitly describing semantics. Approaches like the Semantic Web (Berners-Lee, Hendler & Lassila, 2001) and OWL-S (OWL Services Coalition, 2003) are investigating techniques for externally describing the semantics of content and Web services. This leads to a separation between semantic and syntactical description, which

Figure 3. Service oriented interaction pattern



is beneficial since it does not require consensus to discover that two services perform the same or similar tasks, for example. This approach does, however, introduce the possibility of syntactic mismatches if service discovery is based solely on semantics. For instance, a requester may expect a particular service interface for printing but discovers a service with a different one. Approaches such as Ponnenkanti and Fox (2003) address this problem by constructing adapter chains from adapter repositories.

Service Object

The service object implements the service interface, and it is returned by a service provider at the moment a service requester binds to the provider. Service objects are created and released according to a set of policies. In service orientation, service requesters have no knowledge about the policies followed by a service provider when creating service objects during binding. Different service object *creation policies* exist:

- **Shared object:** The service provider creates a single object that is returned to all service requesters when they bind to the provider; the object is consequently shared by all the requesters.

- **Object pool:** The service provider creates a certain number of service objects. A different service object from the pool is returned to every requester as it binds to the provider according to availability. Once a service object is released, it is returned to the pool for reuse. In this situation, service objects are shared by the requesters, but not concurrently, since two requesters that bind to the provider never obtain a reference to the same object simultaneously. This policy is useful, for example, when resources, such as memory, are limited or when service objects represent a physical resource that is limited in quantity, such as a communications port.
- **One object per requester:** A service object is created for each requester. If the same requester is bound to the service provider several times, the requester always obtains the same service object. This policy requires that the service requester be associated with an identifier by the service provider at the time of binding. This policy is useful, for example, when the service requester interacts with a remote service object across multiple method calls (that is, a session) but does not maintain a continuous connection to the service object, such as if communication is done through a connectionless protocol such as HTTP.
- **One object per binding:** A different service object is created each time a service requester is bound to the service provider.

The choice of the object creation policy is important when providing services that are stateful. A stateful service is capable of maintaining state across several method calls by the same client. The *object pool* and *one object per requester* creation policies are adequate for stateful services. The *shared object* creation policy is not a particularly good policy for stateful services, unless the intention is to explicitly share state among all requesters. The *one object per binding* creation policy can be used for stateful services, but this requires that the service requester is aware of the situation and only binds to the service provider once and keeps the returned service object across all interactions with the service. In contrast, all creation policies are adequate for stateless services.

At the end of the interaction between a requester and a service, the requester must release the service object. This step is necessary since the service provider may need to know at which time the service object is not being used anymore to either destroy the service object or to give it to another requester. Two different release policies can exist:

- **Explicit:** When release is explicit, the service requester explicitly invokes some method that informs the service provider that its interaction with the service has ended.
- **Implicit:** When release is implicit, the end of usage is determined via implicit means such as garbage collection or lease expiration. The concept of leasing allows a service provider to automatically release a service object from a service requester unless the service requester renews the lease. This policy can be used, for example, in parallel with an *object pool* creation policy to guarantee that after a certain amount of time, service objects are released and returned to the pool.

Composition

Service composition represents the usage of a set of services to accomplish a particular task. Service composition is often considered the responsibility of service requesters. In practice, service composition is the incorporation of different services inside of an application to perform some overall function, where the application plays the role of the service requester. Such an application contains control and data flow that coordinates service invocation and data transfer among the different services. The coordinating application can be written in a standard programming language; however, there is a strong tendency to favor the use of executable processes to write such applications. An example of this is BPEL4WS (Andrews et al., 2003), which is used in the context of Web services orchestration (Peltz, 2003).

A service composition is written in terms of service interfaces and is considered abstract until execution time, when service providers are discovered and bound. Service compositions must handle issues relating to service discovery and service dynamics. With respect to service discovery, these issues include service availability, requester-side filtering (see next subsection), and lack of knowledge with regard to service object creation policies. With respect to service dynamics, it is possible that a particular service provider becomes unavailable while the coordinating application is executing. This problem is addressed in Web services through transaction mechanisms that allow rollback in case a service invocation fails. Recovery from a service departure can also be achieved through self-adaptation techniques (Cervantes & Hall, 2003).

Execution

In service orientation, an execution environment provides two main mechanisms to service providers and requesters that support the service-oriented interaction pattern. The first mechanism is service registry access, which includes three main operations:

- **Publish:** Used by service providers to add a service description, along with a reference to the service provider, to the service registry.
- **Remove:** Used by service providers to remove a previously published service description from the service registry. In certain service-oriented technologies, the removal of a service provider from the service registry requires that service objects that the provider has created must be released by the requesters.
- **Lookup:** Used by service requesters to obtain references to service providers present in the service registry. To obtain a service, a service requester sends criteria to the registry that are used to select a set of service providers (*registry-side filtering*); only service providers that match the criteria are returned. The final selection of a specific service provider is left to the requester, which may need to select a single service provider when multiple service providers match the supplied criteria (*requester-side filtering*).

To allow service requesters to be aware about changes in the services, the execution environment provides a second mechanism which is notifications to signal service changes. Through notifications, service requesters can know about changes in service availability to be able to incorporate new services that become available or to stop using services that become unavailable. Service notifications concern the following events:

- **Service published:** An event that occurs when a service is published in the registry.
- **Service removed:** An event that occurs when a service is removed from the registry.
- **Service modified:** An event that occurs when a service that is registered is modified without being removed from the registry. A modification can happen, for example, when the attributes that characterize the service are modified.

When the execution environment does not provide notification mechanisms, service requesters can poll the registry periodically to know when services are published or removed.

A Comparison of Service and Component Orientation

This section summarizes and discusses the similarities and differences between the concepts present in service and component orientation.

Summary

Table 1 summarizes the main similarities and differences that exist between service and component orientation based on the discussion of the preceding sections. From this table, different conclusions can be drawn:

- An important difference between the two approaches is integration time. In component orientation, applications are assembled from building blocks that are integrated at the time of assembly, while in service orientation integration occurs prior to or during execution, since only service descriptions are available during assembly.
- The focus of service orientation is on discovery, while the focus in component orientation is on composition. This explains the fact that service orientation places a stronger emphasis on service description and the separation between service description and implementation.
- Service orientation is concerned with dynamic availability, while this is not the case in component orientation. In general, component orientation is targeted toward the construction of more static applications, where the hypothesis that components

Table 1. Component and service orientation characteristics

	Component Orientation	Service Orientation
Development model	- Building block development separated from assembly. - Assembly based on available components.	- Building block development separated from assembly. - Assembly based on abstract service descriptions.
Building block concept	- External view and implementation not always separated. - Component instances created from components. Components are packaged to support independent delivery and deployment.	- Separation between service description and implementation. - Service providers create service objects. Packaging is not taken into account.
Composition	- Concrete description that defines how a set of component instances are configured and connected together. - Dynamic availability (arrival or departure of components during execution) is not a hypothesis. - Tendency towards structural architecture description.	- Abstract description based on service description. Composition becomes concrete during execution. - Service availability and dynamism need to be taken into account during execution. - Tendency toward use of executable process descriptions.
Execution environment	- Instance creation policy responsibility of the clients. - Life-cycle management through control interfaces. - Non-functional requirements support. - Deployment support.	- Service object creation policy responsibility of service provider and unknown to the requester. - Service registry and notification mechanisms.

may exhibit dynamic availability is not explicitly present, although it may be supported programmatically.

- Service composition favors the use of executable processes to compose services while component orientation favors *structural* architecture description to compose component instances.
- Component orientation gives more responsibility to the execution environment, which covers aspects ranging from low-level deployment to high-level nonfunctional activities. In contrast, service orientation does not explicitly consider low-level activities, such as deployment, and high-level nonfunctional activities are

assumed to be provided by upper layers; for example, transactions are defined at the composition level.

Discussion

It is possible to conclude that service and component orientation are two approaches targeted toward different needs. Service orientation is adequate when the building blocks that form an application exhibit dynamic availability and aspects such as substitutability are important. Component orientation is adequate when applications are assembled from building blocks that are available at the time of application assembly. These two approaches are, however, complementary and can be used together in two different ways.

- **Components as service providers:** Components are ideally suited to be service providers. This approach allows aspects which are not considered in service orientation, such as delivery and deployment, to be taken into account. This approach is already followed by component models such as EJB, where certain components can implement services accessible through the Web.
- **Introduction of service-oriented concepts into component models:** A different approach is to introduce service-oriented concepts into component models. In particular, the service-oriented interaction pattern could be used as a means to connect component instances, which act as service providers and requesters. The benefit of this approach is that it introduces support for late binding and dynamic component availability (that is, the arrival or departure of component instances during execution) to component models. This approach is explored in the service-oriented component model of Cervantes and Hall (2003).

Survey of Service-Oriented Technologies

From the concepts of service orientation previously described, it is possible to establish a list of characteristics that are useful for categorizing *service-oriented platforms*, which are platforms that implement service-oriented principles. The characteristics are:

- **Service description:** the approach for describing services.
- **Service publication:** the operations provided by the service registry so that service providers and requesters can publish and revoke services.
- **Service discovery:** the operations provided to service requesters to discover and bind with service providers as well as registry-side filtering mechanisms.
- **Service object creation policies:** the policies used by a service provider when creating service objects.

- **Service notifications:** the notification mechanisms supported by the platform.
- **Service release:** the policies supported for releasing service objects.

The following subsections use these characteristics to describe the following service-oriented technologies: CORBA Traders, JavaBeans Context, Jini, OSGi framework, and Web services.

CORBA Trader

A CORBA Trader (Stratton, 1998) provides support for the service-oriented interaction pattern in a CORBA environment. The trader belongs to a set of middleware services defined in the CORBA specification (Object Management Group, 1995). The CORBA Trader distinguishes itself from other service-oriented platforms by the fact that it supports the creation of trader networks (called *federations*) that can increase the number of answers that are returned for a service request; these trader networks can continually evolve.

The following is a summary of the CORBA Trader's service-oriented characteristics:

- **Service description:** In the CORBA Trader (Figure 4), a service description is comprised of a reference to a service interface (described in IDL) along with a set of attributes that characterize the service. The number of attributes in the description is fixed, but attributes can be marked as mandatory or optional and also immutable or modifiable. Service descriptions must be published in a service description repository before any service provider can publish provided services of that type to the service registry.
- **Service publication:** The publish and revoke operations are named export and withdraw, respectively. Service attributes are supplied when a service is published.
- **Service discovery:** The discovery operation is named query. This method supports complex requests through a constraint over the properties declared in the service description along with preferences that allow the ordering of responses from the registry and policies targeted towards limiting the propagation of a request in a trader federation. There is no explicit bind operation since the results returned by the service registry contain references to the service objects.
- **Service object creation policies:** CORBA makes an explicit difference between the publication of a shared object and the publication of a factory (called *proxies*) that allow different creation policies to be implemented. To support factory registration, a trader must implement a specific interface.
- **Service notifications:** No notifications are defined as part of the trader specification.

Figure 4. CORBA Trader example

a) Service Description

```
interface PrinterService        // service interface
{
  typedef unsigned long JobID;
  JobID print (in string data);
};

service PrinterServiceDescription { // service description
  interface PrinterService;
  mandatory property string building;
  property short floor;
  mandatory property string type;
  mandatory property string language;
  property string name;
};
```

b) Publication

```
Property[] props = new Property[5];
props[0]="Laboratory";
props[1]=(short)3;
props[2]="Color";
props[3]="Postscript";
props[4]="LabPrinter";
String id = reg.export(printer, "PrinterService", props);
```

c) Removal

```
reg.withdraw(id);
```

d) Discovery

```
lookup.query( "PrinterService",
              "((color == 'black') and
                    (language == 'postscript'))", // Constraint
              "min (floor)", // Answer ordering
              policies,
              desiredProps,
              20, // Max answers
              servicerefs, // Results
              refsiterator, limits);
```

e) Binding

```
Offer:
  building = '36'
  color = 'black'
  floor = 2
  language = 'postscript'
  Reference: IOR:0000000002449444c3a6f6d672e6f7 ...
```

No binding operation since the result includes a reference to a remote object

- **Service release:** The service requester release the service object explicitly through the release method.

JavaBeans Context

The concept of JavaBeans Context was introduced in a subsequent specification to the original JavaBeans component model (Sun Microsystems, 1998). This concept provides a means to group JavaBeans component instances into execution contexts (which can themselves be organized hierarchically) and to allow instances to obtain services from the context at run time.

The application domain for this platform concerns nondistributed applications that are assembled visually and are user oriented (meaning that they normally support interaction through a user interface). In JavaBeans Context, only one service provider for each service can be present in the context (that is, the service registry) at any given moment.

The following is a summary of JavaBeans Context's service-oriented characteristics:

- **Service description:** In JavaBeans Context (Figure 5), a service is described as a Java class or interface. Since only one service provider per service can be registered, there is no support for properties that allow service providers to be differentiated.
- **Service publication:** The publish and revoke operations are named addService and revokeService, respectively. When a service is published, only the name of the service and a reference to the service provider is submitted to the service registry.
- **Service discovery:** JavaBeans Context offers an operation, called hasService, to allow a service requester to test for the availability of a service and an operation, called getService, to bind a service requester with the service provider. During binding, a client can give initialization parameters to the service and, in addition, the service requester is automatically registered to receive notifications concerning the departure of the service.
- **Service object creation policies:** A service provider must implement a method called getService that receives, among other things, a reference to the service requester. This allows different creation policies to be implemented.
- **Service notifications:** JavaBeans Context supports the registration of listeners that receive events announcing either the arrival (serviceAvailable) or departure of a service (serviceRevoked).
- **Service release:** Service requesters must free the service objects explicitly.

Figure 5. JavaBeans Context example

a) Service Description

```
interface PrinterService
{
 public long print(String data);
};
```

b) Publication

```
BeanContextServiceProvider provider = new
PostscriptPrinterProvider();

beancontext.addService(PrinterService.class,printerprovider);
```

c) Removal

```
boolean revokeNow = true; // must service be freed immediately?

beancontext.revokeService(PrinterService.class, printerprovider,
revokeNow);
```

d) Discovery

```
//tester présence du service
if(beancontext.hasService(PrinterService.class)==true) {
```

e) Binding

```
Object service = beancontext.getService(
                 child, //bean instance to service request
                 child, //service requester
                 PrinterService.class,
                 paramsConfig, //configuration parameters
                 child //listener to service removal events
  );
 ((PrinterService)service).print(data);
}
```

Jini

Jini (Arnold et al., 1999) is a distributed service platform defined by Sun that shares several concepts with the CORBA Trader platform. Jini is a Java technology that leverages the capability of the Java platform to dynamically load code from the network. Thanks to this, a service object is sent to the same location as the service requester, although distribution is possible if the object received by the requester plays the role of a *proxy*. This characteristic differentiates Jini from the CORBA Traders where communication between service requesters and the service object is always done remotely. Jini explicitly supports lease policies for service publication and removal. Another characteristic of Jini is that service requesters and providers must initially locate a registry to be able to initiate service discovery. A registry may be known from a fixed address, or

Figure 6. Jini example

a) Service Description

```
interface PrinterService extends Remote
{
 public long print(String data) throws RemoteException;
};
```

b) Publication

```
ServiceRegistrar reg = findRegistry();   // find the registry

Entry entries[]={printerType,resolution,...}; // attributes

ServiceItem printsvc = new ServiceItem(
                        null,         // service id
                        serviceObject,// serializable object
                        entries);     // attributes
ServiceRegistration svcreg = reg.register(registration,
               1000000); // lease time (ms)
```

c) Removal

Lease expiration or:

```
Lease lease = svcreg.getLease();
lease.cancel();
```

d) Discovery

```
Class svcInterfaces []={PrinterService.class};
Entry entries[]={printerType};

ServiceTemplate template = new ServiceTemplate(null, svcInterfaces,
entries);
ServiceMatches matches = reg.lookup(template,3); // 3 max
```

e) Binding

```
if(matches.totalMatches>0)
{
  ((PrinterService)matches.items[0]).print(data);
}
```

it can be discovered from a request that is broadcast. Jini services are organized into *groups,* and a particular service registry can contain a particular group of services. Even though Jini supports the existence of multiple service registries, it does not offer mechanisms that allow registries to delegate service requests; instead, service providers must publish their service offers into multiple registries.

The following is a summary of Jini's service-oriented characteristics:

- **Service description:** In Jini (Figure 6), a service is described as a Java interface with an arbitrary number of attributes, which are subclasses of the Entry class.
- **Service publication:** Service publishing is done through the register method. This

method puts the service in the service registry which grants a lease for the registration. Before the lease expires, the service provider must renew it to avoid the removal of its service from the service registry. To revoke a service that has been published, the service provider can wait for the lease to expire, or it can force its early expiration.

- **Service discovery:** Service discovery is done through a method named lookup that receives a maximum number of answers that the registry should return. Jini does not support sophisticated registry-side filtering of service providers. The criteria used to determine if a service matches with a request is that the service interfaces match with those that were requested and that the attributes sent by the requester are present and equal to those in the service description.
- **Service object creation policies:** During service publication, the service provider includes a reference to the service object and this object is copied into the registry. When a requester obtains an answer from the registry, it also obtains a copy of the service object. By default, the creation policy is *one object per binding*; however, if the service object plays the role of a proxy for a remote object, the creation policy becomes *shared object* since all of the service requesters interact with the same remote object.
- **Service notifications:** Jini provides an asynchronous notification mechanism to inform service requesters about service events. These events include service publication, revocation, and modification. To receive notifications, a service requester must subscribe to events produced by the registry.
- **Service release:** There is no explicit service object release mechanism; instead, this is accomplished through garbage collection.

OSGi

The Open Services Gateway Initiative (OSGi) (Open Services Gateway Initiative, 2003) is an independent, nonprofit corporation working to define and promote open specifications for the delivery of managed services to networks in homes, cars, and other types of networked environments. The OSGi specification defines a non-distributed Java service platform that provides mechanisms to deploy service providers and requesters inside the platform (called the *framework*). In OSGi, services are delivered and deployed in a logical and physical unit called a *bundle*. Physically, a bundle corresponds to a JAR file that contains code and resources (that is, images, libraries, and so forth); logically, a bundle corresponds to a service provider and/or requester. The framework provides administration mechanisms to install, activate, deactivate, update, and remove physical bundles. The activation or deactivation of a physical bundle results in the activation or deactivation of the corresponding logical bundle. When the logical bundle is active, it can publish or discover services and bind to services provided by other bundles through a service registry provided by the platform. In OSGi, the presence of a service in the service registry dictates the valid lifetime of the service objects; that is, the service objects are considered unusable once the service is removed from the registry.

The following is a summary of OSGi framework's service-oriented characteristics:

- **Service description:** In OSGi (Figure 7), a service is described as a Java class or interface along with a variable number of attributes that are name-value pairs. Although these attributes are not specified inside the interface, they are usually defined in the service interface documentation.
- **Service publication:** The publish operation is called registerService. This method receives the name of a service along with the service object and a dictionary of attributes. When a service is registered, the registry returns a reference that is used to revoke the service from the registry through the unregister operation.

Figure 7. OSGi example

a) Service Description

```
interface PrinterService
{
 public long print(String data);
};
```

b) Publication

```
// service implementation
class PSPrinter implements PrinterService, Configurable
{...}

PrinterService printersvc = new PSPrinter();
Dictionary props = new Dictionary();
props.put("printertype","Postscript");
props.put("color","true");

ServiceRegistration reg = bundlecontext.registerService(
        PrinterService.class.getName(),  // Interface name
        printersvc,                      // Service object
        props);                          // Attributes
```

c) Removal

```
reg.unregister();
```

d) Discovery

```
ServiceReferences refs[]=bundlecontext.getServiceReferences(
         PrinterService.class.getName(),
         "(&(printertype=Postscript)(color=*))" //filter
);

if(refs!=null){
  PrinterService printer
        =(PrinterService)bundlecontext.getService(refs[0]);

  if(printer instanceof Configurable)
  {...configure the service...}

  printer.print(data);
}
```

e) Release

```
bundlecontext.ungetService(refs[0]);
```

- **Service discovery:** Service discovery is done through the getServiceReferences method that returns a set of objects that represents references to service providers. The discovery method receives a string that contains a filter in LDAP (Lightweight Directory Access Protocol) query syntax that allows the registry to perform registry-side filtering based on the attributes provided during publication. Binding with a service provider is done explicitly through a method called getService.
- **Service object creation policies:** OSGi supports two different policies regarding service object creation, *shared object* and *one object per requester*. To implement the latter policy, a ServiceFactory object must be used when registering the service. This object is a factory that is responsible for creating service object instances that are specific to each requester. OSGi considers all requests that originate from the same physical bundle as belonging to the same requester, and the service factory automatically receives an identifier corresponding to the bundle from which the request originated.
- **Service notifications:** OSGi provides a service notification mechanism for service requesters using the typical *listener* approach of Java. During listener registration, an optional filter may be specified to reduce the number of events received. The events that the platform supports are service publication, revocation, and modification.
- **Service release:** Service objects are freed explicitly through a method named ungetService, although the framework will automatically free used services when a bundle (that is, the service requester) is deactivated.

Web Services

According to Andrade and Fiadeiro (2001) and Curbera et al. (2001), Web services emerged out of the need for interaction among heterogeneous applications residing inside different companies. Heterogeneity is not only considered at the implementation language level but also at the level of interaction models, communication protocols, and quality of service.

Web services description is realized in a language called Web Service Description Language (WSDL) (W3C World Wide Web Consortium, 2001). This language, which is XML-based, supports the description of service interfaces, data types, communication transport protocols, and service location. The service registry, called Universal Description Discovery and Integration (UDDI) (UDDI Organization, 2002), supports the publication of service descriptions, called *service types*, along with that of service providers, called *businesses*. UDDI is a distributed service registry in which information is replicated at different sites, and as a consequence, a service provider only needs to publish its services to a single registry. In Web services, service discovery is usually carried out by a human and dynamic availability is not as prevalent as in the service platforms presented previously.

The following is a summary of Web service's service-oriented characteristics.

- **Service description:** In Web services (Figure 8), a service description contains the following information:

  definitions: service name and namespace

  types: definitions of complex data types

  message: description of a message (request or response). This description contains the name of the message and a number of parts that describe parameters or return values

  portType: method description that combines several messages, for example a request and a response

  binding: description of the message transmission protocol.

  service: location of the service (as a URI).

- **Service publication:** The main methods that are provided by UDDI to publish information are:

Figure 8. Web services example

a) Service Description

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
   targetNamespace=
   "http://www.foo.com/wsdl/PrinterService.wsdl" xmlns=...>
   <message name="PrintRequest">
      <part name="data" type="xsd:string"/>
   </message>
   <message name="PrintResponse">
      <part name="jobID" type="xsd:string"/>
   </message>
   <portType name="Print_PortType">
      <operation name="print">
         <input message="tns:PrintRequest"/>
         <output message="tns:PrintResponse"/>
      </operation>
   </portType>
   <binding name="Print_Binding"
            type="tns:Print_PortType">
     ...
   </binding>
   <service name="Print_Service">
      <documentation>
        WSDL pour service impression
      </documentation>
      <port binding="tns:Print_Binding" name="Print_Port">
         <soap:address
           location="http://www.printhost:8080/printsvca"/>
      </port>
   </service>
</definitions>
```

b) Publication

```
save_service (example not given for lack of space)
```

c) Discovery and binding

```
find_service (example not given for lack of space)
```

save_service: publish a service type.

save_business: publish a service provider.

The main methods that allow information to be removed from the registry are:

delete_service: remove a service type.

delete_business: remove a service provider.

- **Service discovery:** UDDI provides different discovery methods:

  find_service: returns information about services provided by a business.

  find_business: returns information about one or more service providers. Discovery methods support queries through regular expressions. These methods return keys that can be further used to obtain extended information and values can be returned in an ordered way following different criteria (e.g., alphabetical order, registration date, and certificate availability).

- **Service object creation policies:** All creation policies can be supported by a service provider.
- **Service notifications:** UDDI can notify clients about changes in the registry concerning service types and businesses; notifications include addition, removal, and modification.
- **Service release:** Since web services support different communication protocols both release policies can be implemented by the service providers.

Summary of Service-Oriented Technologies

This section surveyed several technologies that support service-oriented concepts, a summary of which is presented in Table 2. As the technologies surveyed in this section target particular application domains, the purpose of the survey was not to compare these technologies with each other, but to illustrate how service-oriented concepts are realized in the different platforms. The CORBA Trader is useful in CORBA environments for decoupling distributed clients and services. JavaBeans Context is useful to introduce service-oriented concepts to centralized user-oriented applications. Jini is useful for providing and discovering services in *ad hoc* networks, where the service may or may not be remote. OSGi is useful as a remotely administrable gateway for dynamically deployable services or as a framework for building non-distributed service-oriented applications. Finally, Web services are useful for providing functionality that is accessible via Web-based protocols.

Conclusion

This chapter presented an overview of service-oriented concepts, a comparison of service to component orientation, and a survey of a set of service-oriented technologies.

Table 2. Characteristics summary

	CORBA Trader	JavaBeans Context	Jini	OSGi	Web Services
Service description	IDL interface + mandatory and optional properties	Java interface or class	Java interface + attributes	Java interface or class + attributes	WSDL description
Service publishing	export withdraw	`AddService` `revokeService`	`Register` `lease.cancel` or lease expiration	`RegisterService` `unregister`	`save_service` `delete_service`
Service discovery and results filtering policies	`Query` constraint language, result ordering, policies	`GetService` no filtering	`Lookup` attributes in the request must be present in service description	`GetService References` / `getService` LDAP filter	`find_service`
Service object creation policies	shared object or all (if proxy)	all	an object per binding or shared object	shared object or an object per requester	all
Notifications	not defined	service arrival and departure	arrival, departure, modification	arrival, departure, modification	arrival, departure, modification
Release	explicit	explicit	implicit using both garbage collection and leasing	explicit	both
Distributed	yes	no	yes	no	yes
Number of registries	multiple registries that collaborate and form a federation	one registry per context, but can be grouped hierarchically	multiple non-collaborating registries	one registry	multiple replicated registries
Other characteristics		only one service provider per service can exist in a given context	service objects are downloaded to the client's location	support for deployment of service providers and requesters	interaction spans across a long period

Service orientation is an approach for building computing systems around an interaction pattern where a client is bound to a server that is unknown to the client until execution time. In contrast, component orientation is a software development approach that focuses on components as software building blocks that are explicitly assembled into different applications. These two approaches are similar since both services and components are used as building blocks to construct applications.

The technology survey covered a set of service-oriented technologies that support the interaction pattern associated with the service-oriented approach. These technologies target different kinds of application domains, ranging from nondistributed constrained environments (in the case of OSGi) to distributed heterogeneous applications (in the case of Web services). The variety of domains in which service orientation is used reflects a more general desire of software developers to defer selection of building blocks until run time and/or accounts for the possibility of dynamically available building blocks.

Although service orientation started as an approach based around a particular interaction pattern, it is slowly evolving into a full-blown software development approach, thanks in part to the popularity of Web services. Service orientation introduces, however, new challenges to software development, such as discovering services, handling dynamic availability, and requester-side filtering. Currently, service discovery is mostly done based on syntactic approaches; the addition of semantic information would enable more sophisticated ways to perform this activity. Handling dynamic availability requires providing means to applications so that they can be capable of releasing departing services or incorporating new services during execution. Requester-side filtering requires the definition of criteria that allows the selection, among multiple providers of a same service, of the candidate that is most appropriate.

References

Andrade, L. F., & Fiadeiro, J. L. (2001). *Coordination technologies for Web-services.* In OOPSLA 2001: Workshop on Object-Oriented Web Services. Retrieved August 3, 2004: *http://www.research.ibm.com/people/b/bth/OOWS2001/andrade.pdf*

Andrews, T. et al. (2003). Business process execution language for Web services, version 1.1. Retrieved August 3, 2004 : *http://www-106.ibm.com/developerworks/library/ws-bpel/*

Arnold, K., O'Sullivan, B., Scheifler, R. W., Waldo, J., & Wolrath, A. (1999). *The Jini specification*. Reading, MA: Addison-Wesley.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American, 284*(5), 34-43.

Bieber, G., & Carpenter, J. (2001). Introduction to service-oriented programming (rev 2.1). Retrieved August 3, 2004 : *http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf*

Box, D. (1998). *Essential COM.* Boston: Addison-Wesley.

Burbeck, S. (2000). The evolution of Web applications into service-oriented components with Web services. Retrieved August 3, 2004 : *http://www-106.ibm.com/developerworks/library/ws-tao/index.html*

Cervantes, H., & Hall, R. S. (2003). *Automating service dependency management in a service-oriented component model.* Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction.

Retrieved August 3, 2004: *http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6/Proceedings/papersfinal/p28.pdf*

Clements, P. C. (1996). *A survey of architecture description languages.* Proceedings of the 8th International Workshop on Software Specification and Design. IEEE Computer Society (pp. 16-25). Washington, D.C.

Conan, D., Putrycz, E., Farcet, N., & DeMiguel, M. (2001). *Integration of non-functional properties in containers.* Proceedings of the Sixth International Workshop on Component-Oriented Programming. Retrieved August 3, 2004: *http://research.microsoft.com/users/cszypers/events/WCOP2001/ConanPutryczFarcetDeMiguel.pdf*

Curbera, F., Nagy, W. A., & Weerawarana, S. (2001). *Web services: Why and how.* In OOPSLA 2001: Workshop on Object-Oriented Web Services. Retrieved August 3, 2004: *http://www.research.ibm.com/people/b/bth/OOWS2001/nagy.pdf*

Davidson, M. (2002). The Bean Builder tutorial. Retrieved August 3, 2004: *http://java.sun.com/products/javabeans/beanbuilder/1.0/docs/guide/tutorial.html*

Fowler, M. (2004). Inversion of control containers and the dependency injection pattern. Retrieved August 3, 2004: *http://martinfowler.com/articles/injection.html*

Indulska, J., Bearman, M., & Raymond, K. (1993). *A type management system for an ODP trader.* Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing ICODP (pp. 141-152). Berlin, Germany.

Meijler, T., & Nierstrasz, O. (1997). Beyond objects: Components. In M. P. Papazoglou, & G. Schlageter (Eds.), *Cooperative information systems: Trends and directions* (pp. 49-78). London: Academic Press.

Object Management Group. (2003). *CORBA components, V3.0*. Retrieved August 3, 2004: *http://www.omg.org/technology/documents/formal/components.htm*

Object Management Group. (1995). The common object request broker: Architecture and specification. Retrieved August 3, 2004: *http://www.omg.org/technology/documents/formal/corba_2.htm*

Open Services Gateway Initiative (2003, March). OSGi service platform specification, 3rd release. Retrieved August 3, 2004: *http://osgi.org/resources/spec_download.asp*

Ousterhout, J. K. (1998). Scripting: Higher-level programming for the 21st century. *Computer*, 31(3), 23-30.

OWL Services Coalition. (2003). *OWL-S: Semantic markup for Web services.* Retrieved August, 3, 2004: *http://www.daml.org/services/owl-s/1.0/owl-s.html*

Peltz, C. (2003). Web services orchestration and choreography. *Computer, 36*(10), 46-52.

Ponnenkanti, S. R., & Fox, A. (2003, March). *Application-service interoperation without standardized interfaces.* Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom) (pp. 30-40). Fort Worth, TX.

Stratton, D. (1998). *The OMG CORBA Trader service* (Tech. Rep.). University of Ballarat, Australia, School of Information Technology and Mathematical Sciences.

Sun Microsystems. (2001). Enterprise JavaBeans specification version 2.0. Retrieved August 3, 2004: *http://java.sun.com/products/ejb/docs.html*

Sun Microsystems. (1998). Extensible runtime containment and services protocol for JavaBeans version 1.0. Retrieved August 3, 2004: *http://java.sun.com/products/javabeans/glasgow/beancontext.pdf*

Sun Microsystems. (1997). Java Beans specification. Retrieved August 3, 2004: *http://java.sun.com/products/javabeans/reference/api/index.html*

Szyperski, C. (1998). *Component software: Beyond object-oriented programming*. Boston: Addison-Wesley.

UDDI Organization. (2002). UDDI version 3.0 specification. Retrieved August 3, 2004: *http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3*

W3C World Wide Web Consortium. (2001). Web services description language (WSDL) 1.1. Retrieved August 3, 2003: *http://www.w3.org/TR/wsdl*

Chapter II

Beyond Application-Oriented Software Engineering: Service-Oriented Software Engineering

Jiehan Zhou
VTT Technical Research Centre of Finland, Embedded Software, Finland

Eila Niemelä
VTT Technical Research Centre of Finland, Embedded Software, Finland

Abstract

This chapter introduces SOSE (Service-Oriented Software Engineering) as an advanced software development. It argues that SOSE is characterized by small projects, existing software reuse, market changing and software evolution focusing, customer domination, and common middards in comparison with AOSE (Application-Oriented Software Engineering). It presents SOSE software development methodology involving the main processes of service extracting, service middard, service circulation, service evaluation, and service evolution with the middard service fundamental. Eventually, compared with other industries (for example, car manufacturing, construction, and electronics) with global standards and fine-granularity components, the software industry is immature in unified service standards, service marketplace, and service granularity

evaluation. The authors hope that understanding the underlying fundamental SOSE middard service and SOSE methodology will make the software industry more productive and profitable.

Introduction

As the number of component services (for example, ActiveXs, DCOMs, and CORBAs) grows, e-business software development is coming into being. Concretely, over-engineered systems with redundant functionality are not required for the majority of customers. Software organizations are typically of a small size, in a state of continual process change, never arriving, and always in transition (Bennett et al., 2000). Software developers prefer exploiting the services available in the marketplace to produce the most effective software in the least time rather than programming from scratch. If needed, software will be produced as a particular service, instead of "a system," conforming to a service standard technology. The system could be composed, executed, maintained, and evaluated in the way of online service procuring, engaging, and changing.

Currently, almost all commercial application software is sold on the basis of ownership (Bennett et al., 2002). Thus, a customer buys the object code with some form of license to use it. Any updates, however important to the customer, are the responsibility of the vendor. Any attempt by the customer to modify the application is likely to invalidate warranties as well as ongoing support. This form of marketing, known as supply-side, is facing the following challenges:

- Bringing together users and providers of software in a trusted marketplace.
- The continuously changing software market and customer needs. Today's software development is in the way of e-business, in which customers are expecting and demanding various timely services from sites, not costly and time-consuming turnkey products.
- Reducing software development cost and time. Supply on customers' demand is one of the most successful ways to reduce software development cost and time.
- Large-scale and complex software systems. The systems we need to build are likely to get more complex. Making service standards or specifications enables us to successfully develop large complex software systems.
- Evolution in Internet time. This challenge is to achieve very fast change yet provide very high quality software. Existing software maintenance processes are simply too slow to meet the needs of much faster implementation of software changes.

In recognition of these challenges, studies have been running all over the world, aiming at developing new approaches to software development for highly agile software systems, which design, implement, test, evaluate, and access services across the Web.

Table 1. Differences between AOSE and SOSE

Application-oriented software development	Service-oriented software development
• Supply-side method	• Demand-side method
• Product	• Instant service
• System	• Particular one when needed
• Ownership	• Loose-coupled
• Rigid boundaries	• Unfixed boundaries
• Technology first	• Non-technology first
• Large-size organization (spindle-shape)	• Small-size organization (dumbbell-shape)
• Several months	• Procurement first
• Product of a calculator	• Service of multiplying

The emerging key concept is that software is a service rather than an application. The differences between AOSE development and SOSE development are shown in Table 1.

An application view is one, typically dominated by suppliers, in which suppliers are closely coupled with customers' business problems and software solutions; customers buy and own the application, product, or system offered by suppliers. The supply-side methods, driven by technological advance, work well for systems with rigid boundaries of concern such as embedded systems. It can also benefit from being a large-size spindle-shaped organization, which focuses on product development, not the product market and maintenance. The nature of the application-oriented mode implies a slower time-to-market and high cost associated with the maintenance and evolution of the application.

By contrast, a service view is one, typically dominated by customers, in which an application is broken down into smaller, finer grained parts; organizations are with the characteristics of small-size and dumbbell-shaped (software market and maintenance-focusing); customers have no interest in owning the whole application but use the parts as they require. This implies that application functionality is delivered as a service where functionality is required and service elements are identified, executed, and then discarded. That is so-called instance service. The demand-side methods are driven by nontechnology issues, such as supply contracts, terms, and conditions.

In fact, a study of the demand-side mode has been developed and applied early in manufacturing (Iacocca Institute, 1991), such as agile manufacturing, in which a new car might be produced on a dynamic product supply chain. The nodes in the chain, small-size companies, work together in the form of rapid service request/service response. This enhances companies accelerating the time-to-market of a product, reducing the investment risk, and rapidly responding to the market change. Similarly, with the growth of well-structured building blocks and service standards, there exists advanced software development models in software industry, such as COTS (commercial off-the-shelf) model (Garlan, Allen & Ockerbloom, 1995) and C-BSE (component-based software engineering) model (Pree, 1997; Szyperski, 1998). What is the difference between them and SOSE? What is the SOSE model?

This chapter attempts to answer these questions is organized as follows. The first section presents the concepts related to SOSE, followed by a section comparing SOSE with the

existing advanced software development methods. The fourth section develops the SOSE model and illustrates its key elements. The challenges in SOSE are presented in the final section.

Notions and Visions in SOSE

Notions in SOSE

The widely accepted definition of a service is: "An act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production" (Lovelock, Vandermerwe & Lewis, 1996). A service represents a self-contained Internet-based application, capable of completing tasks on its own and able to discover and engage other services to complete higher-level transactions. A service is something that you find, use as and when needed and then discard (Bennett et al., 2000; Brereton et al., 1999).

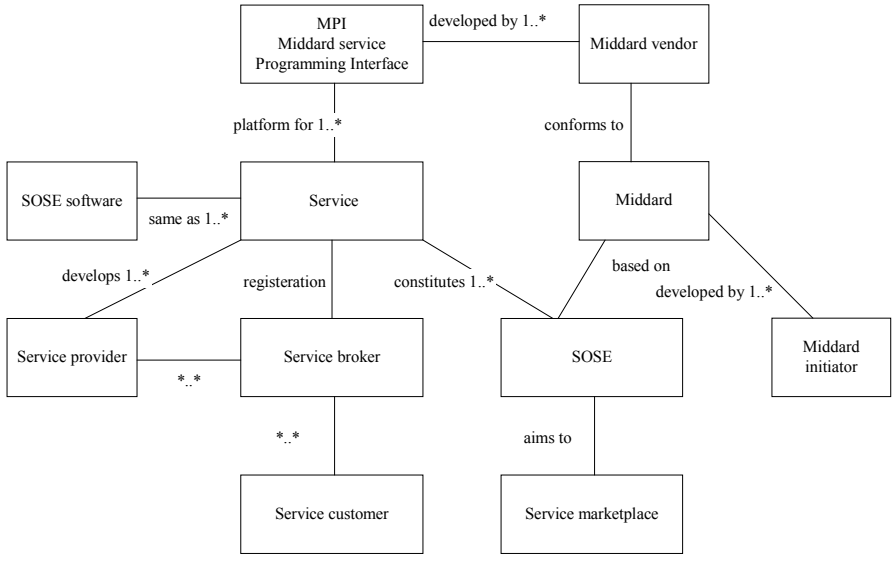
In summary, a service is a public program conforming to a middard standard, developed by service providers, traded by service brokers, and used and accessed when service customers need it. Note that a service might be an HTML text, a CORBA (Common Object Request Broker Architecture) object, or an ODBC (open database connectivity method) server. They are all compliant to a middard, such as a programming guideline, specification, or standard. Middard is derived from the connotation of the words of middleware and standard. Many literatures specify standards, like CORBA (Object Management Group, 1996), ODBC (Universities and Colleges Software Group, 1995), and TPM (Transaction Processing Monitor) (The Open Group, 1992), as middleware technologies. This chapter introduces middard as a term of generalizing the above various specifications, which essentially provide mediations or standards to enhance the communication among multicomputers, multipeople, and multidomains.

SOSE breaks an application down into several services, in which services are developed and configured independently and separately in conformance with a middard. This strategy allows service providers to focus on the business problems at hand, independently developing and managing a service to meet needs at a specific point in time. The conceptual SOSE model is shown in Figure 1.

SOSE software is developed or assembled by a dynamic organization group, based on existing services with the characteristics of conformance with a middard, business problem focus, and minimum programming and customization. SOSE software is sometimes called SOSE service.

Middard is a documented agreement containing specifications to be used consistently as standards, specifications, methods, rules, and guidelines for developing, describing, and managing services sharable to software stakeholders. A middard is usually initiated, developed, and managed by middard initiator. TPM, CORBA, HTML, and XML are instances of middards. A middard is also referred to as a service middard.

Figure 1. Conceptual model of SOSE description



A middard vendor is any supporter who implements either a part or whole of a middard and packages them into a middard service programming interface (MSPI) for service providers to facilitate service development. A middard may have multiple middard vendors. A service provider, also called a service supplier, develops and maintains middard-based SOSE services.

Service brokers are those who act as intermediaries, responding to the requests from service customers and service providers to make SOSE services circulation easier in the service marketplace. Service customers, also called service requestors/users, are those who acquire, select, and use SOSE services.

A service profile contains acceptable values for contract terms and policies to govern how these values may be negotiated. A service contract contains the terms agreed on by the service provider and service customer for the supply of the service. A customer profile contains acceptable legal systems for contracts, the minimum service performance required, the maximum acceptable cost, and the percentage of average market cost within which negotiation is possible. Provider profiles contain acceptable legal systems for contracts, guaranteed performance levels, and the cost of providing the service.

Comparison in Advanced Software Development

This section compares the AOSE method with the SOSE method using the items of the software market, development objective, software organization, provider-customer relationship, and middard and software risk. The summary is shown in Table 2.

Table 2. Comparison between AOSE and SOSE

		Market	Organization	Objectives	Customer Involvement	Relationship	Middard	Risk
AOSE		Single-single	Spindle-shaped	Application	Close-coupled	Provider-dominated	Miscellaneous	Lower
SOSE	COTS-BSE	Many-many	Dumbbell-shaped	COTS	Expertise-maintaining	Long-term partnership	Provider-side	Low
	C-BSE	Many-many	Dumbbell-shaped	Component	Loose-coupled	Customer - broker-provider	Consistent	High
	VE-BSE	Many-many	Dynamical	Organization	Loose-coupled	Customer-sponsor	Common	Higher

Application-oriented software development is a typical waterfall mode, in which a software organization is responsible for a product or system owned by a customer. Thus, a spindle-shaped organization focuses on software products.The organization consists of various teams involving requirement analysis, application design, application coding, testing and integrating, and application delivery. They sequentially work together on a thorough consideration of customer business logic, techniques, and market changes during the product life cycle. AOSE customers are close-coupled with application providers and middard vendors. The middard used in AOSE varies between different middard vendors. AOSE is slow to respond to market and customer variations, time consuming, and costly.

COTS-BSE (COTS-based software engineering), C-BSE (component-based software engineering), and VE-BSE (virtual enterprise-based software engineering) are the representatives of SOSE. If such an organization emphasizes new product research and product marketing, it is dumbbell-shaped. If an organization focuses on changing markets and dynamic project teams, the organization is dynamical. COTS-BSE particularly uses COTS products as elements of the COTS-BSE software, due to shrinking budgets, accelerating rates of COTS enhancement, and expanding system requirements (Tran & Liu, 1997). COTS-BSE is a procurement-centric method, in which customers directly buy COTSs from providers. As in the case of procurement, the customer needs to maintain expertise and processes for technology evaluation to be able to identify and assess alternative or additional future providers. Meanwhile, the provider may also control product evolution with the result that it becomes very difficult for the customer to move to another provider. So, COTS-BSE software is based on a provider-side middard. The main activities in CBSE are COTS identification, evaluation, and integration. Establishing criteria for COTS evaluation is vital for realizing COTS-BSE. However, COTS-BSE has a risk of architecture mismatching.

Software components can be considered to be units of independent production, acquisition, and deployment that interact to form a functional system (Poulin, 2001; Syzperski, 1998). C-BSE refers to the development of software systems from pre-existing parts. C-BSE aims to create platform-independent component integration frameworks, which provide standard interfaces and thus enable flexible binding of encapsulated software components. In C-BSE, providers are able to register their components with a broker and

thus make information about their products available to potential customers. Customers are supported by the broker in making trade-offs between their own requirements and the offerings of providers, so the relationship between customers and providers is loose-coupled. C-BSE components have to be associated with a consistent middard. C-BSE software is high risk since customers may not be able to find a replacement if a component ceases to be available or if a provider goes out of business.

A virtual enterprise is a temporary consortium or alliance (that is, so-called VE-BSE sponsor) of organizations formed to share costs and skills and exploit fast-changing market opportunity (Walton & Whicker, 1996). The VE-BSE sponsor consists of a series of co-operating "nodes" of core competence which form into a supply chain to address a specific opportunity in the marketplace. Each node is a VE-BSE organization. VE-BSE organizations do not produce complete products in isolated facilities. They operate as service nodes in a network of providers, customers, engineers, and other specialized service functions. VE-BSE will materialize by selecting skill services and asset services from different firms, synthesizing them into a single business entity. VE-BSE goes through four distinct activities in conformance with a common middard: organization identification, organization formation, organization operation, and organization termination. In each of them, decision processes, such as organization evaluation and selection, operation redesign, and organization termination, are involved and sequentially related (Mahajan, 1995).

COTSs, components, and VEs are all service entities and compliant to a certain middard. VE is the extremity of SOSE organizations. SOSE is a revolutionary activity emerging well beyond the application-oriented paradigms that preceded it. SOSE are in a commonality of breaking down a large application or system into units. SOSE allows service providers to develop and manage these units independently and simultaneously. Therefore, SOSE is concerned less with building parts than providing users with constantly reliable parts that maintain continuously working software. In contrast to application-oriented software development, SOSE has the following advantages:

Quick solution: SOSE focuses on providing the user with a solution rather than a product. That is, SOSE emphasizes the analysis of specific users' requirements and service marketplace. If there are well-defined services for use, SOSE conducts the service procurement. For services that cannot be found, SOSE allows consumers to post a notice for service providers to respond to. Therefore, SOSE organizations are normally small-sized and dumbbell-shaped, focusing on market changes and customers' business.

Making up for insufficient resources: It is impractical for a software organization to build everything every time. Lack of sufficient money or personnel are usually the main causes for software system development failure. SOSE makes it possible for software development to explore and use services available in the marketplace but cannot develop them costly and timely. There is some competitive know-how, which is difficult for an individual provider to master in a short time. Therefore, SOSE makes it cheaper for an organization to cooperate with a competent partner to provide users with a quick solution.

Facilitating large-scale and complex software development: A complex system usually involves contributors from different domains, for example, computer-aided airplane

manufacturing software. Many problems will arise from the development and maintenance of a complex software system without a unified communication standard. SOSE provides this unified standard, one kind of a middard, which makes the integration in different domains easier.

Productivity: The real productivity benefits of SOSE will be achieved by enabling parallel service development. As the adoption of building system from standard components taken by automobile manufacturing, electronic, construction, and many industries, SOSE will be a new paradigm in the maturing software industry, which is more productive and profitable.

SOSE Vision

This section illustrates SOSE from the viewpoints of service customers, service providers, service brokers, and service organizations. SOSE vision is shown in Figure 2.

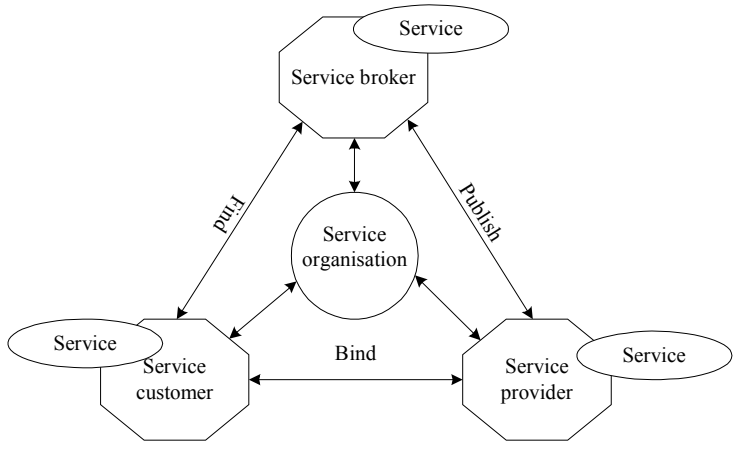
Customers' viewpoint: Software is a service, not an application. Services are easy to be customized by customers themselves. Customers obtain the copyright of using services through contract negotiation. In this way, it is necessary for customers to change the attitude of possessing software into using software.

Providers' viewpoint: Software is a service unit that can be used independently but are more likely to be used as a module in an integrated system. The provider is responsible for managing the module within its life cycle (for example, module designing, implementing, and maintaining). SOSE providers obtain service requirements from service marketplaces, not from customers directly. Services are designed in conformance with a certain middard. After implementing services, providers submit the detailed information of the services (for example, functionality, nonfunctional characteristics, and business information) to service brokers rather than to customers.

Brokers' viewpoint: Software is a service to be registered and sold. Brokers provide a number of facilities to respond to requests from both service providers and service customers. Facilities to support SOSE may include those for ranking and selecting candidate services, visualization of services and of their closeness to fit requirements, and automated support for certification. Brokers share benefits from selling services with service providers.

Software organizations' viewpoint: The global software marketplace is coming into being. The knowledge (for example, user requirements, technologies, expertise, programmers, and partnerships) corresponding with software development is highly dynamic and changeable. For instance, the users change their requirements; employees leave; new technology emerges; and partners join. It is imperative that software organizations use an agile development method that allows flexibility and accommodates change. Unlike application-oriented software development, based on rigid business boundaries, spindle-shaped organization structure, and time-consuming software ownership delivery, SOSE organizations take into account the activities of service purchase, organization alliance, and middard selection during the initial phase of software development. SOSE organizations mainly focus on customers' business rather than the supporting technologies in the phase of software implementation. SOSE organizations emphasize improving their

Figure 2. SOSE vision



key competence and putting more effort toward service qualities in the phase of software maintenance. In this way, software organizations will be more customer-centered, small-scaled, agile, and competitive.

SOSE Conceptual Model

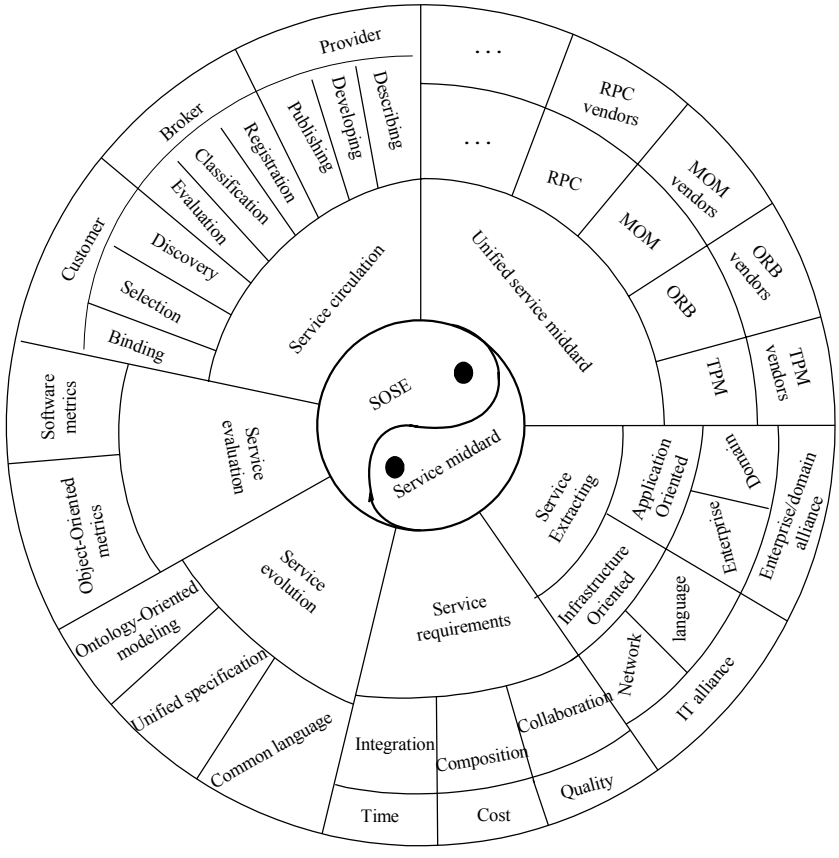
SOSE Model

The SOSE conceptual model is shown in Figure 3. SOSE software is composed of interactive and interoperative services. SOSE requirements are not only from the traditional time-to-market (cost and quality) but also from enterprise application integration, system composition, and collaboration across platforms, in which service providers do not need to know who service customers are. Service organizations and providers discover the commonality underlying the pre-existing software, operation systems, and SOSE marketplace. This process is also called service extraction. Service middards provide service providers with the framework for describing, designing, publishing, and assembling services. Service customers use a service in a binding way when needed, which is called *bind once, execute once.*

Service Extracting

Service extraction packages the general classes into one service and lets the surrounding parts in the original classes use the newly packaged service. Generally, there is a possible reusable part in an existing software. SOSE service extracting aims at specifying this part

Figure 3. SOSE conceptual model



for service sharing. SOSE service extracting includes application-oriented service extracting and infrastructure-oriented service extracting. Application-oriented service extracting is a process of extracting common services from existing applications. Application-oriented service extracting can be divided into extracting enterprise services and domain services.

- Extracting enterprise services. Amounts of modules are developed repeatedly in enterprise applications due to the factors of technologies and platforms. The key issue for extracting enterprise services is to extract common services in enterprise applications as enterprise services, which will enhance the efficiency of new system development.
- The key issue for extracting domain services is to extract common services in a domain as the domain services.

Infrastructure-oriented service extracting can be divided into extracting network services and extracting programming language services.

- The key issue for extracting network services is to discover common services, which are responsible for connecting various kinds of network devices. These services include multiprotocols transformation, virtual serving, automatic load balancing, and fault hiding. They need not only to transfer right data to the end user but also guarantee the QoS of network, network and information security (Adachi, Kikuchi & Katsuyama, 2000).
- The key issue for extracting programming language services is to discover common language services, which implement data configuration, object class running, and components interoperation across platforms (Silberschatz, Korth & Sudarshan, 1997).

Service Middards

Service middards provide technological support for the implementation of the extracted service. There are various service middards that have been widely used for various purposes. Table 3 summarizes the initiation, implementation, and evolution of well-known service middards. They are also the representatives of various standardized solutions.

- **Transaction Processing Monitor (TPM):** A transaction is a complete unit of work. It may comprise many computational tasks, which may include user interface, data retrieval, and communications. A typical transaction modifies shared resources (X/Open, 1992). TPM was initially developed as multithreaded servers to support numerous terminal transaction requests from a single process. It improves batch and time-sharing application effectiveness by creating online support to share application services and information resources. ACID (Atomicity, Consistency, Isolation and Durability) is the basic requirements for TPM. The core services defined in TPM are (Silberschatz et al., 1997): presentation facilities to simplify creating user interfaces, persistent queuing of client requests and server responses, routing of client messages to servers, and the coordination of two-phase commit when transactions access multiple servers. Some commercial TPM vendors are CICS from IBM, Top End from NCR, and Encina from Transarc (Houston, 1998). TPM technology is widely used for delivery order processing, hotel and airline reservations, electronic fund transfers, security trading, and manufacturing resource planning and control. ODBC and distributed transaction processing monitor (DTPM) extend TPM services. ODBC contains ODBC API and ODBC SQL grammar, which enables any transaction application to communicate with any database manager. DTPM extends TPM services (for example, identification and authorization), which enable completing global transactions.

Table 3. Summary of well-known service middards

Name	Initiation	APIs	Vendors	Relative terms	Evolution
TPM	Transaction processing	x_reg, x_prepare, x_commit, x_rollback, ….	IBM's CICS, Microsoft's MTS, NCR's TopEnd …	Many terminals Single-Mainframe, 2-tier C/S, N-tier C/S, database application,…	ODBC, DTPM
ORB	Object interworking	X_remoteObjectDelegate, x-_StubDelegate,…	IONA's ORBIX, SunSoft's NEO IBM's DSOM…	CORBA, COM/DCOM, …	Inter-ORB(Java/RMI)
MOM	Loosely-coupled connection, Message queuing	MQConn, MQOpen, MQClose,…	IBM's MQSeries, Microsoft's MSMQ, Java Messaging Service (JMS),…	Asynchronous mechanism, event-driven applications, message queuing, publish/subscribe,…	Combination with ORB, MOM across Intranet and Internet
RPC	Interprocess call	x_binding, x_naming, x_UUID, …	ONC's RPC, X/Open's DCE RPC, CORBA IIOP,…	Synchronous mechanism,	Object-oriented, TCP/IP support

- **Object Request Broker (ORB):** ORB is initially developed as a middleware technology that facilitates application integration across different programming languages, hardware platforms, operating systems, and ORB implementation (Object Management Group, 1996). ORB applications are composed of objects, which are all identical in functionality. The core services needed in ORB are interface definition, location and possible activation of remote objects, and communication between clients and objects. Two major ORB initiatives are the CORBA specification from OMG and COM from Microsoft. Their services are similar but slightly different. There are a number of commercial ORB products available, such as ORBIX by IONA, NEO by SunSoft, and DSOM by IBM. One trend for ORB is to specify a set of APIs that can be implemented in different ORB products (for example, Java/RMI). Another trend continues toward intranet- and Internet-based applications.
- **Message-Oriented Middleware (MOM):** MOM provides an assured, asynchronous, and connectionless method to exchange messages between processes (Houston, 1998). MOM commonly satisfies these important conditions: no simultaneous connection is required between the message sender and receiver; there are extremely strong request and response delivery guarantees even when communication does not occur simultaneously between the sender and receiver; requests and responses can be translated and reformatted en route between senders and receivers. MOM may be more suitable for wide-area and large-scale systems. MOM has a larger share of the market than ORB. There are many MOM products from different vendors. For example, there are the IBM MQ Series, Microsoft MSMQ, and the Java Messaging Service (JMS). MOM is being designed towards a combination with ORB, for example, IBM's D-Sphere (Tai, Mikalsen, Rouvellou & Sutton, 2001), MOM across intranet and Internet, for instance, SunTM ONE Middleware (Sun Microsystems, 2003), and MOM across vendors, such as MSMQ-MQSeries Bridge (Microsoft Corporation, 2003).

- **Remote Procedure Call (RPC):** RPC is a type of protocol that allows an application client on one computer to execute an application server on a server computer. RPC is similar to the local procedure call in that one thread of control logically winds through two processes (Sun Microsystems, 1998a). The RPC protocol makes no restrictions on the concurrency model implemented. For example, an implementation may choose to have RPC calls to be asynchronous so that the client may do useful work while waiting for the reply from the server. Another possibility is to have the server create a task to process an incoming request so that the server can be free to receive other requests. The RPC protocol must offer the following services: the unique specification of a procedure to be called, matching response messages to request messages, and authenticating the caller to service and vice versa. RPC is appropriate for communications between applications that require a short response times and relatively small amounts of data transfer. The major RPC products are ONC RPC (Sun Microsystems, 1998b), X/Open DCE RPC (The Open Group, 1997) and CORBA IIOP (Soley, 1992). RPC is being specified to contain network protocols, security service, and object management for supporting Intranet- and Internet-based computing.

Additionally, there are so many middards emerging for service describing, discovering, and integration, such as XML (eXtensible Markup Language) and UDDI (Universal Description, Discovery and Integration).

Service Circulation

Service circulation plays an important role in supplying the right SOSE services for the right SOSE users at the right time. Service circulation includes service providing, service brokering, and service executing. Service providing spreads providers' profiles to the outside in order to bring customers, other providers, and partners of services. The key issues for service providing are as follows:

- **Service mining for market opportunities:** The key business for providers is shifting from application development to market tracking and maintenance of existing services.
- **Description of service profile:** This describes meta information held by services for contract negotiation, including terms of functionality, performance, cost, and provider profile.
- **Implementation of services:** Services are implemented using a middard platform (for example, HTML, DCOM, or CORBA platform) provided by middard vendors, which allows service providers to focus on customers' business logic rather than supporting technology.
- **Publication of services:** Implemented services can be published to a service broker and thus information about services is available to potential customers.

Service brokering offers the services of classifying, evaluating, and registering services. The key issues for service brokering are as follows:

- **Classification of services:** Classification of services is needed to help customers find what they really want. Text mining techniques in knowledge discovery area will be helpful for service classification and management.
- **Registration of services:** In the process, each service must be assigned a Universal Unique IDentifier (UUID) for guaranteeing uniqueness.
- **Ranking of services:** This ranks and visualizes services by service profiles and provider profiles for customer selection. An evaluation criterion is exactly needed to be made for automation service rating.
- **Query of services:** Query service in service brokering enables customers to obtain execution and configuration information for the different services.

Service executing enhances services to customers by dynamically combining and executing the services in a just-in-time way. The key issues for service executing are as follows:

- **Discovery of services:** Service discovery provides a wide range of choice array that meets customers' needs. In contrast to service classification, service discovery performs for customers. In order to improve searching efficiency, it is necessary for three parties (service provider, broker, and customer) to comply with a same classification criterion.
- **Selection of services:** Referring to the ranking service result provided by service brokering, customers make a selection from their profiles.
- **Binding of services:** In this phase, customers and providers connect dynamically, and services are executed as needed. At the extreme, the binding that takes place prior to execution is disengaged immediately after execution to permit the SOSE software to evolve for the next execution.

Realizing the importance of service circulation, the United States and the United Kingdom have established the service circulation environment, based on the vendors of Visual Basic and Java, respectively Flashline (Flashline, 2003) and ComponentSource (ComponentSource, 2003). Flashline is one of the service brokers taking Java components (Beans or Enterprise Beans) and .NET/COM components as the services. Services offered by Flashline consist of service registering, categorizing, and listing. ComponentSource takes Microsoft components (COM, VBA) and Sun Microsystems' components as the services and evaluates them by the items of installation/uninstallation, antivirus, and description of services.

Table 4. One of SOSE service performance metrics

Metrics	Meaning
Existence of meta information	If the value is 1, users of components can easily understand components' usage that the components' developers assume.
Rate of component observability	The ratio of the number of readable properties to the number of attributes.
Rate of component customizability	The ratio of the number of registered properties to the number of attributes.
Self-completeness of component's return value	The ratio of the number of business methods without return value to the number of business methods.
Self-completeness of component's parameter	The ratio of the number of business methods without parameters to the number of business methods.

Service Evaluation

Service evaluation is the business of rating service performances and qualities. High-quality services normally bring high-quality SOSE systems. One or two low-quality services will remarkably decrease the overall system qualities. Consequently, it is necessary to measure services before using them. In addition, object-oriented is the basis of composing SOSE services recently. Some methods have been studied on measuring software (ISO/IEC, 1991) and object-oriented programs (Chidamber & Kemerer, 1994). These methods are also helpful for measuring SOSE services. Service evaluation includes performance and quality measurement. Much effort has been devoted to defining and describing the metrics involved in measuring service performance and quality (Poulin, 2001; Washizaki, Yamamoto & Fukazawa, 2002). One of SOSE service performance metrics is shown in Table 4. One of SOSE service quality metrics is shown in Table 5.

In service evaluation, there is one empirical method worthy to be recommended. That is *take-try-and-use,* which is widely accepted and applied by service providers, brokers, and customers today. In this, the provider commonly offers customers a free using period, and customers make a contract with providers after the customer is satisfied with the service. Take-try-and-use enables customers to make a wide range of options. Moreover, take-try-and-use is high risk for long-term and critical SOSE software, so service brokers may get added value through a third party for a wider and commercial service scale, like certification, pricing/licensing information, and so forth.

Table 5. One of SOSE service quality metrics

Metrics	Meaning
Productivity	The ratio of total development hours for the project to total lines of code contained in the components that make up the product.
Reliability	The amount of errors which are figured out.
Stability	The ratio of the total number of open change requests to the total number of requirements.
Reuse	The ratio of the reused lines of code to the total lines of code.

Service Management and Evolution

Service management provides a number of facilities to support three parties (service customers, service providers, and service brokers). Service customers expect the service of shorter qualification interval, lower qualification cost, faster service delivery, service variations, and lower risk to be offered. Service providers expect to get the information on service needs, customers' feedback, timely service middards, and potential partners. Service brokers expect to get information on the latest service middards, service classification, service retrieval, service evaluation, and service adaptation. Concretely, service management includes the following functions:

- **Service classification:** Service specification, users' feedback, service evaluation, service rating, and service listing.
- **Service selection:** Service description, service commerce properties, service providers' trustworthiness.
- **Service evolution:** Service development, service maintenance, and service extraction.

In recent years, software evolution, recognized as an essential aspect of software systems, has become an emerging research subject. In service evolution, a service business modification and a service middard modification are roughly considered as two key activities. Service business modification extends the capabilities and functionality of a service to meet the further needs of the service customer, possibly in major ways. The process of a service middard modification consists of the following activities: modification analysis, modification implementation, modification review/acceptance, and application for international standard. The objective modifying an existing service middard is to preserve its higher user base. Some intelligent mechanisms may be needed to manage the dynamic and complex service evolution, such as semantics-based service modeling and higher-level service middard making.

Table 6. SOSE supporting status

SOSE requirements	Supporting status
Service circulation	Already exists
Service evaluation	Not properly supported
Unified service middards	Diversification and competition
Unified middard vendors	Not properly supported
Service extracting	Some results in knowledge discovery
Service evolution	To be researched

Discussion

SOSE business takes a service as a unit that will be individually maintained, updated, and reused within the life cycle of SOSE software. SOSE is also oriented to software innovation. SOSE enables SOSE providers to pay more attention to service business and evolution by making a service choice before starting a new SOSE software development. SOSE providers focus on selecting a service middard and implementing customers' business with awareness of what other providers are doing. SOSE organizations work on core competitive services with a small team. It is important for SOSE organizations to have quick response to market changes and service evolution rather than delivering a large two-to-three years-consuming system. SOSE allows customers to take-try-and-use a service cheaply when needed.

SOSE results from the continually maturing software market and continuously growing standardization technology and software customization. SOSE has a close association with traditional software development methods. There are many methods and techniques appearing in traditional software development and other mature industries (for example, car manufacturing, construction, and electronics) available for SOSE implementation. For instance, e-business product circulation has many applications. More service middards are specified and accepted by providers, vendors, and customers. However, it is the beginning for service evolution research. Table 6 states SOSE supporting status briefly.

The study of the service middard and service evolution will be emphasized in SOSE. Future research issues involved in SOSE also include the following topics:

- **Extended object-oriented service modeling:** Object-oriented is the basis for implementing SOSE. Objects have been criticized for their lack of emphasis on semantics. Due to the public service characteristic of SOSE running on unanticipated platforms, domains, sites, and cultures, object-oriented will be limited. Therefore, it is necessary to develop or apply an extended object-oriented modeling with more strict semantic constraints for SOSE. For instance, ontology-oriented modeling will be an optional one (Ikeda, 1997).

- **Unified service middard:** Different service middards, developed by different initiators, limit the range of service usage. A unified service middard will be helpful in service description, requisition expression, service decomposition, service implementation, and service combination. Certainly, it is support from service providers, brokers, and customers that make the unified service middard meaningful.
- **Unanticipated requirement discovery:** Unanticipated requirement is a distinct characteristic for SOSE. The conventional requirement with being easily attainable, explicit, big piece, and customer-visiting no longer exists in the continually maturing software market. SOSE organizations face implicit requirements from a potential customer group rather than an individual customer. Service customers are able to make various options for their needs. Another key issue is to discover implicit requirement. An existing code's value adding and organization alliance will be helpful to SOSE organization survival. Adding an existing code's value causes service life to lengthen by building the legacy code as a service, extending the service, and promoting the service quality and performance. An organization alliance is another possible option available for SOSE organization. One activity for SOSE requirement discovery is to find more partners for existing services.
- **Automated engagement:** Automated engagement enables customers and providers to operate one transaction of service providing and service execution over the Internet automatically. Automated engagement allows customers to get what is really needed with just one mouse action. The level of customer participation in software development is reduced in AOSE, COTS-BSE, C-BSE, and VE-BSE. Automated engagement involves service allocating, service engaging, service persisting, service rolling back, and so forth. Meanwhile, automated engagement must be based on a highly unified service middard and a strict unified service evaluation method.
- **Multidisciplinary and interdisciplinary research:** Agile manufacturing is a mainstream advanced manufacturing method in the manufacturing industry, in which study emphasizing the combination of human-factor, technology-factor, and organization-factor has been done. Similarly, SOSE, as an agile software development in the software industry, is nontechnology dominated with a focus on software innovation and providing what customers really want. That implies that SOSE software will be more human-centric with maximum customers and easy customization, so SOSE software engineers are required to have the knowledge of service psychology, service negotiation policy, marketing management, social behavior area, and so forth. Therefore, how to reform the existing software engineering discipline is another important topic for SOSE.

Conclusion

We introduced the concept of an SOSE model as an advanced software development, which will make the software industry more productive and profitable. Current compo-

nent-based software development methods (for example, COTS-BSE, C-BSE, and VE-BSE) are SOSEs. SOSE differentiates into AOSE small projects, existing software reuse, market changing and software evolution focusing, customer domination, and common middards. Customers may access SOSE software as data of a public service repository when needed. SOSE software can be shared by multitudinous systems due to its compliance with the common middard.

We presented the basic concepts and principles of SOSE middard services. Middard services define standards for common service description, implementation, management, and discovery. Additionally, a SOSE software development model involving the main processes of service extracting, service middard, service circulation, service evaluation, and service evolution was presented. Compared with other industries (for example, car manufacturing, construction, and electronics) with global standards and fine-granularity components, the software industry is immature in unified service standards, service marketplace, and service granularity evaluation. Typically, coordination among multiple service middards becomes the bottleneck of wide service usage. Eventually, the main challenges in SOSE, such as extended object-oriented service modeling, a unified service middard, unanticipated requirement discovery, automated engagement, and multidisciplinary and interdisciplinary research, was addressed.

References

Adachi, M., Kikuchi, S., & Katsuyama, T. (2000, November 20-24). *NEPRI: Available bandwidth measurement in IP networks.* Proceedings of 7th IEEE Singapore International Conference on Communication Systems (pp. 511-515).

Bennett, K. H., Gold, N. E., Munro, M., Xu, J., Layzell, P. J., Budgen, D., et al. (2002). Prototype implementations of an architectural model for service-based flexible software. *Proceedings of 35th Hawaii International Conference on System Sciences, 8,* 76-85.

Bennett, K. H., Layzell, P. J., Budgen, D., Brereton, P., Macaulay, L., & Munro, M. (2000, December 5-8). *Service-based software: The future for flexible software.* Proceedings of the 7th Asia-Pacific Software Engineering Conference (pp. 214-221), Singapore.

Brereton, P., Budgen, D., Bennett, K., Munro, M., Layzell, P., Macaulay, L., et al. (1999). The future of software: Defining the research agenda. *Communications of ACM, 42*(12), 78-84.

Chidamber, S., & Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Transaction on Software Engineering, 20*(6), 476-493.

ComponentSource. (2003). About ComponentSource. Retrieved August, 5, 2004: *http://www.componentsource.com/*

Flashline. (2003). Whitepapers. Retrieved August 5, 2004: *http://www.flashline.com/*

Garlan, D., Allen, R., & Ockerbloom, J. (1995, April 23-30). *Architectural mismatch: Or why it's hard to build systems out of existing parts.* Proceedings of the 17th International Conference on Software Engineering (pp. 179-185), Seattle, WA.

Houston, P. (1998). Building distributed applications with message queuing middleware. Retrieved August 5, 2004: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmqqc/html/bldappmq.asp*

Ikeda, K. S. (1997, August 23-29). *Task ontology makes it easier to use authoring tools.* Proceedings of the 15th International Joint Conference on Artificial Intelligence (pp. 342-347), Nagoya, Japan.

ISO/IEC. (1991). *ISO/IEC 9126 International standard: Information technology - Software product evaluation - Quality characteristics and guidelines for their use*. International Standard ISO/IEC 9126.

Kontio, J. (1996, March 25-30). *A case study in applying a systematic method for COTS selection.* Proceedings of the International Conference on Software Engineering (pp. 201-209), Berlin.

Lovelock, C., Vandermerwe, S., & Lewis, B. (1996). *Services marketing* (3rd ed.). London: Prentice Hall International.

Mahajan, R. (1995, March 7-9). *Building the virtual enterprise.* Proceedings of the 4th Annual Agility Forum Conference (pp. 32-40), Atlanta.

Microsoft Corporation. (2003). Chapter 22: Administration and management of MSMQ-MQSeries bridge. *Host integration server 2000 resource kit.* Retrieved August 5, 2004: *http://www.microsoft.com/resources/documentation/host/2000/all/reskit/en-us/part4/hisrkc22.mspx*

Object Management Group. (1996). *The common object request broker architecture and specification* (2nd ed.). Boston: John Wiley & Sons.

The Open Group. (1997). DCE 1.1: Remote procedure call. Retrieved August 6, 2004: *http://www.opengroup.org/products/publications/catalog/c706.htm*

The Open Group. (1992). Distributed TP: The XA specification. Retrieved August 6, 2005: *http://www.opengroup.org/bookstore/catalog/c193.htm*

Poulin, J. S. (2001). Measurement and metrics for software components. In G. T. Heineman, & W. T. Councill (Eds.), *Component based software engineering: Putting the pieces together* (pp. 435-452). Boston: Addison-Wesley.

Pree, W. (1997, December, 2-5). *Component-based software development - A new paradigm in software engineering.* Proceedings of the Joint Asia-Pacific Software Engineering Conference and International Computer Science Conference (pp. 523-524), Hong Kong.

Preiss, K., & Goldman, S. (Eds.). (1991). *21st century manufacturing enterprise strategy*. Bethlehem, PA: Lehigh University.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (1997). *Database system concepts.* Boston: McGraw-Hill.

Soley, R. M. (Ed.). (1992). *Object management architecture guide: OMG TC document 92.11.1* (2nd ed.). New York: John Wiley & Sons.

Sun Microsystems. (2003). Sun ONE middleware. Retrieved August 5, 2004: *http://wwws.sun.com/software/product_family/middleware.html*

Sun Microsystems, Network Working Group. (1988, April). RFC 1050 RPC: Remote procedure call protocol specification. Retrieved August 5, 2004: *http://www.faqs.org/rfcs/rfc1050.html*

Sun Microsystems, Network Working Group. (1988, June). RFC 1057 - Remote procedure call protocol specification: Version 2. Retrieved August 5, 2004: *http://www.faqs.org/rfcs/rfc1057.html*

Szyperski, C. (1998). *Component software: Beyond object-oriented programming.* Boston: Addison-Wesley.

Tai, S., Mikalsen, T. A., Rouvellou, I., & Sutton, S. M. (2001, September 4-7). *Dependency-spheres: A global transaction context for distributed objects and messages.* Proceedings of the 5th International Enterprise Distributed Object Computing Conference, Seattle, WA.

Tran, V., & Liu, D. B. (1997, January 13-16). *A risk-mitigating model for the development of reliable and maintainable large-scale Commercial-Off-The-Shelf integrated software systems.* Proceedings of the International Annual Reliability and Maintainability Symposium on Product Quality and Integrity (pp. 361-367), Philadelphia, PA.

Universities and Colleges Software Group. (1995, September). Delivering data to the desktop: ODBC overview. Retrieved August 5, 2004: *http://www.liv.ac.uk/middleware/html/overview.html*

Walton, J., & Whicker, L. (1996). Virtual enterprise: Myth and reality. *Journal of Control*, 22(8), 22-25.

Washizaki, H., Yamamoto, Y., & Fukazawa, Y. (2002, October 3-4). *Software component metrics and its experimental evaluation.* Proceedings of International Symposium on Empirical Software Engineering, Rome, Italy.

Chapter III

Service Composition: Concepts, Techniques, Tools and Trends

Boualem Benatallah
University of New South Wales, Australia

Remco M. Dijkman
University of Twente, The Netherlands

Marlon Dumas
Queensland University of Technology, Australia

Zakaria Maamar
Zayed University, United Arab Emirates

Abstract

This chapter provides an overview of the area of service composition. It does so by introducing a generic architecture for service composition and using this architecture to discuss some salient concepts and techniques. The architecture is also used as a framework for providing a critical view into a number of languages, standardization efforts, and tools related to service composition emanating both from academia and industry and to classify them in terms of the concepts and techniques that they incorporate or support (for example, orchestration and dynamic service selection). Finally, the chapter discusses some trends in service-oriented software systems engineering pertaining to service composition.

Introduction

The last decade has seen organizations worldwide expose their operations on the Web to take advantage of the commoditized infrastructure and the potential for global visibility and increased business process automation that Web technologies offer. An overwhelming number of organizations have reaped the benefits of the Web by making their applications available to their customers and partners through interactive interfaces combining Web forms and dynamically generated Web pages. This has seen the Web evolve from a vehicle for information dissemination to a vehicle for conducting business transactions, albeit in a manual way.

The next step in the evolution of Web technologies is the emergence of Web services (Alonso, Casati, Kuno & Machiraju, 2003). Web services bring together ideas from Web applications on the one hand (for example, communication via document exchange) and distributed computing on the other hand (for example, remote procedure calls and communication middleware). The outcome of this convergence is a technology that enables applications to communicate with each other in a programmatic way through standardized message exchanges. This is expected to trigger a move from a Web of mostly manual interactions to a Web of both manual and programmatic interactions.

There are several definitions of Web services, most of which agree on saying that a Web service is a software application available on the Web (through a URI) whose capabilities and modus operandi are described in XML and is able to communicate through XML messages over an Internet transport protocol. At present, a widely accepted core infrastructure for Web services is the so-called Web Services Stack which is essentially structured around three XML-based standards: SOAP, WSDL, and UDDI (Curbera, Duftler, Khalaf, Nagy, Mukhi & Weerawarana, 2002). These three standards are intended to support the tasks of service description, discovery, and communication.

This basic core infrastructure is currently being used to build simple Web services such as those providing information search capabilities to an open audience (for example, stock quotes, search engine queries, auction monitoring). However, it has rapidly become clear that this core infrastructure is not sufficient to meet the requirements of complex applications (especially in the area of B2B integration) since it lacks abstractions for dealing with key requirements, such as security, reliability, transactions, composition, service level agreements, and quality of service, among others (Medjahed, Benatallah, Bouguettaya, Ngu & Elmagarmid, 2003). In light of this, several efforts are underway to design a standard comprehensive infrastructure for Web services.

In particular, the development of new services through the composition of existing ones has gained considerable momentum as a means to integrate heterogeneous enterprise applications and to realize B2B e-commerce collaborations. Unfortunately, given that individual services are developed using manifold approaches and technologies, connecting and coordinating them in order to build integrated services is delicate, time-consuming, and error-prone, requiring a considerable amount of low-level programming and system administration efforts. This observation has sparked a wave of R&D efforts in an area often known as “service composition”.

Stated in simple terms, service composition aims at providing effective and efficient means for creating, running, adapting, and maintaining services that rely on other services in some way. In order for service composition to deliver on its promises, there is a need for development tools incorporating high-level abstractions for facilitating, or even automating, the tasks associated with service composition. Hence, these tools should provide the infrastructure for enabling the design and execution of composite services.

This chapter provides an overview of the benefits and pitfalls of service composition, the functionalities that the supporting platforms are required to provide, and the extent to which these requirements are addressed by the current state of the art. However, the chapter will not address in detail system issues such as reliability, transactions, and security. We present the main concepts for service composition by presenting a generic tool architecture for service composition, covering aspects such as design of composite services and composite service execution. Based on these concepts, we provide a survey of service composition models, methods, and supporting technologies.

The chapter is structured as follows: The Generic Architecture section discusses the foundation concepts in Web services composition by introducing a generic tool architecture for service composition. The Languages for Service Composition section overviews language support for Web services description and composition, covering the design module of the generic architecture. The Platforms for Composite Service Execution section reviews research efforts and commercial platforms for web services composition by covering the runtime module of the generic architecture. The Trends Relevant to (Web) Service Composition section reviews some trends in Web services technologies, and the last section provides concluding remarks.

Generic Architecture

From an architectural point of view, a tool environment for service composition should provide at least the following modules:

- **Design module:** This module offers a graphical user interface for specifying composite services. The module may also support translation of a composite service design into a description language. More advanced design tools may support the automated verification and/or simulation of composite service designs on the basis of a formal language.
- **Runtime environment:** This module is responsible for executing a composite service and routing messages between its components. It is also responsible for monitoring and fault and exception handling. The runtime environment may additionally support dynamic service selection and binding as discussed below.

Figure 1. Generic architecture for a service composition tool

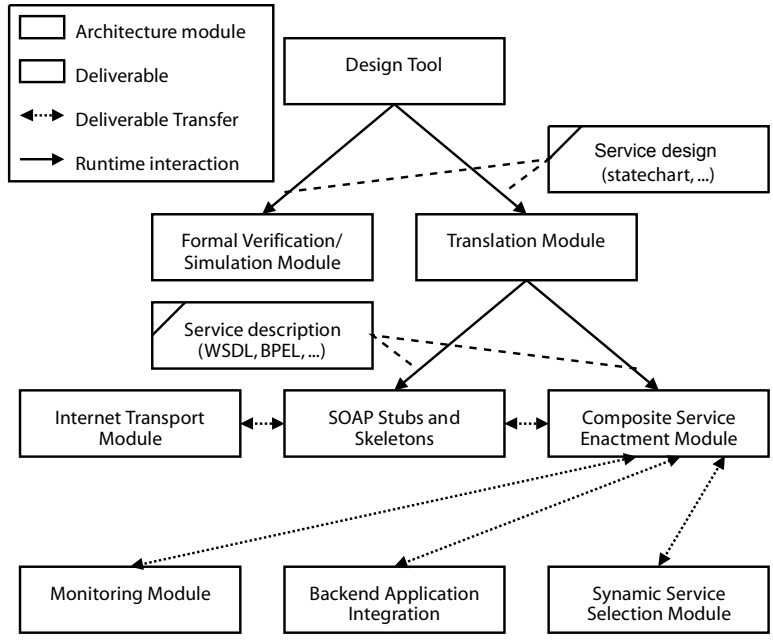


Figure 1 represents the generic architecture in more detail. This section explains the generic architecture further.

Composite Service Design

A tool for composite service design supports a composite service design methodology. A methodology consists of design languages, formalisms that are coupled with these design languages, and design approaches. Design languages are graphical notations that can be used by stakeholders in a design process to represent a design from their perspective. They focus on representing a service composition in a way that is easy to understand for the stakeholders. Formalisms are mathematical languages that can be used to represent a particular aspect of a design. As a mathematical language, a formalism provides a mathematical basis for verification and simulation of a design. In a composite service, for example, a formalism provides techniques that allow designers to analyze if two services can be composed. An overview of formalisms that are used in model-driven service composition is given in the Languages for Service Composition section. A design approach prescribes a series of steps that have to be taken to construct a design. In this way, a design approach provides a structured way to construct a design by gradually introducing more detail into user requirements and current business operations until a

level of detail is reached at which a design can be directly implemented. For an approach to service composition, this is the level of detail at which a design in a one-to-one fashion corresponds to a description that can be executed by a runtime environment. Such a description is a textual (typically XML-based) representation of the functional and nonfunctional properties of a service or service composition. The Languages for Service Composition section, Description Languages subsection explains some existing techniques for describing service compositions.

From existing description techniques in the area of service composition, we can derive that there are currently two ways to design a service composition, namely *choreography* or *orchestration*. A choreography differs from an orchestration with respect to where the logic that controls the interactions between the services involved should reside.

A choreography describes a collaboration between some enterprise services to achieve a common goal. Hence, it does not focus on a particular service but rather on a goal. Therefore, the control logic is distributed over the involved services and the choreography emerges as the services interact with each other. To design a choreography, we first describe the interactions that enterprise services have with each other to achieve their goal and then the relations that exist between these interactions. A choreography does not describe the actions that are performed internally by the service providers to realize their enterprise services. Figure 2 shows a typical example of a choreography. This example shows a collaboration that relates to buying an item.

An orchestration describes the behavior that a service provider implements to realize a service. Hence, it focuses on a particular service, and the control logic is centralized on the service provider of which we implement the behavior. To design an orchestration, we describe the interactions that the service provider has with other parties and the actions that the service provider performs internally to realize the service. An orchestration is meant to be executed by an orchestration engine, as will be explained in the Composite Service Execution subsection. Therefore, it is also called an executable process.

From these observations we can derive a set of basic concepts that are important in the design of service composition, regardless of whether a choreography — or an orchestration-oriented approach is chosen and of the description or design language that is

Figure 2. An example of a choreography

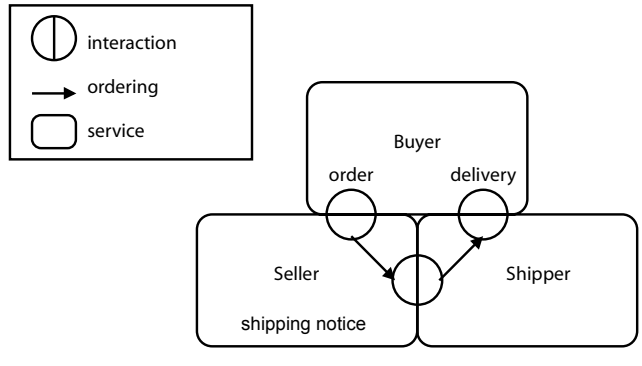
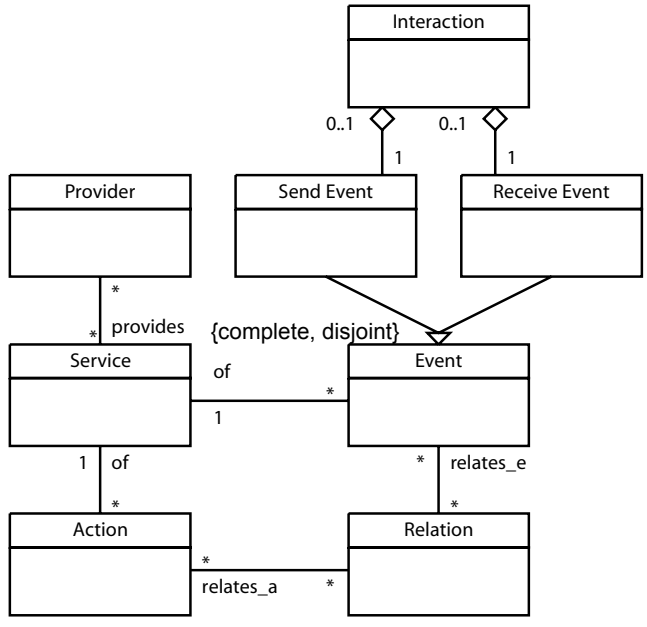


Figure 3. Basic design concepts for service composition design



used. Figure 3 shows a meta-model in which our basic concepts are represented. The figure shows that a service composition consists of a number of services that are provided by service providers. The same service can be provided more than once by different service providers (for example, a flight booking service can be provided by different airlines). A service consists of (internal) actions and events that are part of an interaction with other services. We claim that interactions are based on message passing because this is the basic mechanism for interaction that is used in the mainstream service description languages as they are presented in the Description Languages subsection. Hence, interactions consist of a send event and a receive event. Relations relate actions and interactions to each other. The kind of relation that is used (for example, flow relation, causal relation, state-based relation, and so forth) depends on the language that is used.

Composite Service Execution

The composite service execution engine is the runtime component of a service composition tool. It takes as input a composite service description and coordinates the execution of the composite service according to that description. At least two different execution models can be distinguished:

- **Centralized:** (see, for example, Schuster, Georgakopoulos, Cichocki & Baker, 2000 and Casati & Shan, 2001). In this model, the responsibility for coordinating the execution of a composite service relies on a single "scheduler." This scheduler interacts with each of the component services by processing and dispatching messages. The internal architecture of the central scheduler is similar to that of a traditional workflow management system (van der Aalst & van Hee, 2002), except that the resources are all services rather than human actors, and there is no shared database through which information can be implicitly passed from one stakeholder to another. Instead, information must be explicitly passed through message exchanges.
- **Peer-to-Peer:** (see, for example, Mecella, Parisi-Presicce & Pernici, 2002 and Benatallah, Sheng & Dumas, 2003). In this model, the responsibility for coordinating the executions of a composite service is distributed across the providers of the component services, which interact in a peer-to-peer way without routing messages through a central scheduler. The composite service execution environment therefore manifests itself in the form of a collection of inter-connected modules, which communicate through an agreed protocol. This execution model bears some similarities with distributed workflow execution models, such as those described in Muth, Wodtke, Weissenfels, Dittrich, and Weikum (1998) and Chen and Hsu (2002).

It is crucial that a mechanism is provided for monitoring the executions of a composite service. Indeed, being able to trace the execution of a composite service is crucial for metering, accounting, customer feedback, adaptation, and service improvement. The monitoring mechanism varies depending on the execution model. In the case of centralized execution, the central scheduler can maintain a database of execution traces. In the case of peer-to-peer execution, however, the information about composite service executions is disseminated across a number of distributed data sources hosted by the providers of the component services. Accordingly, it is necessary either to consolidate these distributed data sources periodically or to be able to answer queries on demand (Fauvet, Dumas & Benatallah, 2002).

A composite service can be linked to its component services either in a static or a dynamic manner. A link between a composite service and a component service is static when it is established at design time and cannot be changed without modifying the design of the composite service. A link with a component service is dynamic when a mechanism selects, at runtime, the actual service that will be invoked to perform a given step in the composite service execution. We call this approach *dynamic service selection*.

The pool of candidate services over which the dynamic selection occurs may be: (i) determined at design time; (ii) obtained by evaluating a given query over a registry (for example, UDDI registry); or (iii) obtained from an invocation to a brokering service. The selection itself is then performed based on a set of requirements and using a set of preferences expressed in the composite service description. These constraints and preferences may involve both functional attributes (that is, attributes describing the

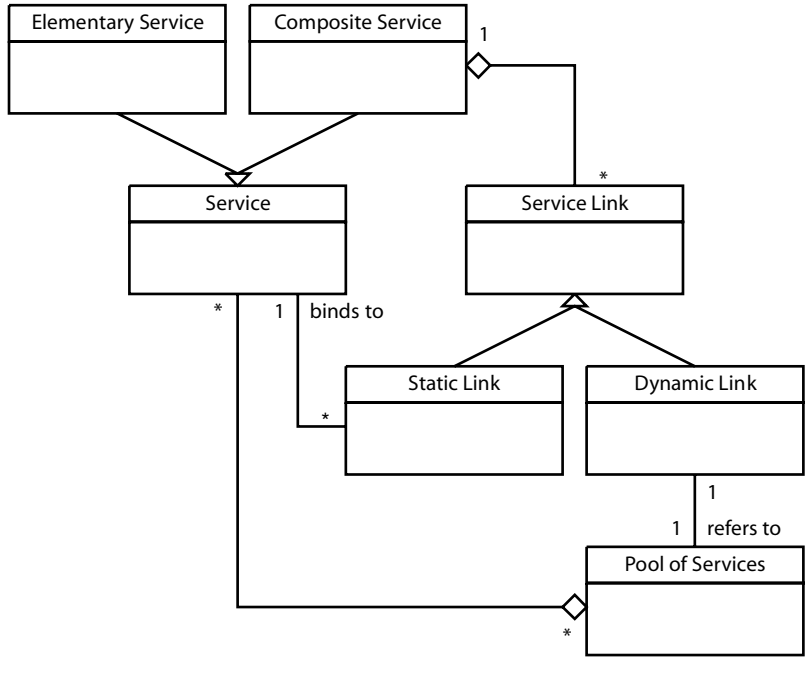
capabilities of the services) and nonfunctional attributes (for example, time, location, price, reliability, trust).

Once a service within the pool of candidate services is selected, it has to be invoked by the composite service. This implies that either all the candidate services for a given task of a composite service offer exactly the same interface (that is, the same set of operations and common constraints on their use) or that some late binding mechanism is used to "homogenize" the interfaces provided by all services so that at the end, the composite service can invoke any of these candidate services.

The CORBA Dynamic Invocation Interface (DII) is an example of a late binding mechanism. Another example from the area of inter-organizational workflow is provided by the CrossFlow system (Grefen, Aberer, Hoffner & Ludwig, 2000). In this system, a task in a workflow can be linked to a contract. When the task needs to be executed, a matchmaking facility attempts to find another workflow that complies with that contract. In the area of Web services, the Web Services Invocation Framework (WSIF) (Apache Web Services Project, 2003) has been developed for the purpose of enabling late binding of Web services.

Figure 4 represents the concepts that are described in this subsection in a meta-model.

Figure 4. Basic execution concepts for service composition design



Languages for Service Composition

The Generic Architecture section explained the need for different types of languages for describing service compositions from different viewpoints. In this section, we present some of these languages. We limit the discussion to description languages and formalisms, leaving aside design languages because there is not yet a widely accepted standard for such languages.

Description Languages

- **BPEL4WS:** The Business Process Execution Language for Web Services (BEA Systems, Microsoft, IBM & SAP, 2003) is a language with an XML-based syntax, supporting the specification of processes that involve operations provided by one or several Web services.

  BPEL4WS is intended to support the description of two types of processes: abstract and executable. An abstract process is a partially ordered set of message exchanges between a service and a client of this service. It describes the *behavioral interface* of a service without revealing its internal behavior. Using the terminology introduced in the previous section, an abstract process is a two-party choreography involving a service provider and a service requestor, described from the perspective of the provider.

  An executable process on the other hand, captures the internal behavior of a service in terms of the messages that it will exchange with other services and a set of internal data manipulation steps. An executable process is composed of a number of constituent activities, the partners involved in these activities, a set of internal variables, and a set of activities for handling faults and transactional rollbacks. Using the terminology of the previous section, a BPEL4WS executable process corresponds to an orchestration specification.

  BPEL4WS draws upon concepts developed in the area of workflow management. When compared to languages supported by existing workflow systems and to related standards (for example, XPDL, WSCI, and ebXML BPSS), it appears that BPEL4WS is relatively expressive (Wohed, van der Aalst, Dumas & ter Hofstede, 2003). In particular, the *pick* construct is not supported in many existing workflow languages. On the negative side, it can be said that BPEL4WS lacks orthogonality, in the sense that it has many constructs with overlapping scope (for example, the *switch* and *sequence* constructs overlap with the *control link* construct).

- **WSCI and BPML:** The Business Process Management initiative (BPMi) is an industry consortium aiming at contributing to the development of (service-oriented) process description standards. The consortium has published a specification for a service-oriented process description language called BPML (Business Process Modeling Language), similar in many ways to BPEL4WS. BPML draws on a previous standard called WSCI (Web Service Conversation Interface) developed by the stakeholders behind BPMi. WSCI integrates many of the

constructs found in BPML and BPEL4WS (for example, sequence, choice, parallel execution, send/receive primitives, and so forth). However, it differs from them in its intent: while BPML is mainly intended for describing orchestrations, WSCI is intended for describing choreographies (see Composite Service Design subsection in the previous section for a discussion on this dichotomy). The strong commonalities between these languages suggest that orchestration and choreography correspond to two different (and complementary) viewpoints over the same class of models (that is, composite service models). As discussed earlier, BPEL4WS has been designed with the goal of capturing both orchestrations and two-party choreographies. Peltz (2003) discusses the relationships between WSCI, BPML, and BPEL4WS in more detail.

- **ebXML BPSS:** Electronic Business XML (ebXML) is a series of standards intended to provide an implementation platform for business-to-business collaborations. ebXML adopts a choreography-based approach to service composition. Specifically, a business collaboration is described as a set of Collaboration Protocol Profiles (CPP) (UN/CEFACT & OASIS, 2001a). A CPP describes, among other things, which part of a given business process a given partner is able to provide by referring to a role in a process specified using the Business Process Specification Schema (BPSS) (UN/CEFACT & OASIS, 2001b). A BPSS document specifies a number of transactions, the roles associated with these transactions, the flow of control and flow of documents between these transactions, and the document access rights for the involved documents. Control-flow relationships are described using guarded transitions (like in state machines) and fork/join operators.
- **WS-CDL:** The W3C Web Service Choreography Description Languages (WS-CDL) (W3C World Wide Web Consortium, 2002) is another ongoing standardization effort in the area of service composition. Like WSCI and ebXML, the intent of WS-CDL is to define a language for describing multiparty interaction scenarios (or choreographies), not necessarily for the purpose of executing them using a central scheduler but rather with the purpose of monitoring them and being able to detect deviations with respect to a given specification.
- **RosettaNet:** RosettaNet (RosettaNet, 2004) is an industry consortium, which has developed a series of standards for Business-to-Business (B2B) integration with an emphasis on supply chain management. Among others, RosettaNet defines a notion of Partner Interface Protocols (PIP), which enables the description of interactions between business processes deployed by multiple partners. The notion of PIP is related to the notion of service choreography and has influenced efforts in this area. For details about RosettaNet and its relationship to Web service standards, readers are referred to Bussler (2003).

Formalisms

In an attempt to provide a rigorous foundation to service composition and to enable the use of formal verification and simulation techniques, a number of formalisms for describing composite services have been proposed. One of the earliest proposals in this

area is that of Cardelli and Davies (1999), who present an algebra for programming applications that access multiple Web resources (also called *services*). This algebra brings together operators inspired by process algebras (sequential execution, concurrent execution, and repetition) with operators capturing the unreliable nature of the Web (timeout, time limit, rate limit, stall, and fail). Basic services are described using the operator *url*, which attempts to fetch the resource associated with a given URL. Although the algebra is intended for manipulating Web pages, it could conceivably be extended to take into account the richer structural and behavioral descriptions of Web services.

Various authors have advocated the use of Petri nets as a formal foundation for modeling composite services or for defining formal semantics for service composition languages. The VISPO project (Mecella et al., 2002) has advocated the use of Petri nets to model the control flow aspects of composite services. Van der Aalst (2003) examines a number of proposed standards for service composition in terms of a collection of workflow patterns and notes that these proposed standards would benefit from having a formal semantics defined in terms of established formalisms such as Petri nets. Finally, Narayanan and McIlraith (2002) present DAML-S, a language that supports the description of composite services, and defines a mapping from the process-oriented subset of DAML-S to Petri nets.

More recently, Bultan, Fu, Hull, and Su (2003) adopt Mealy machines (a category of communicating automata with queues) to describe the interactions (also called *conversations*) between aggregated services. Each service participating in an aggregation is described as a Mealy machine, which consumes events from a queue and dispatches events to the queues of the other services in the aggregation. The authors study the expressive power of the resulting formalism, measured in terms of the set of traces (that is, sequences of events) that can be recognized by an aggregation of Mealy machines.

There is not yet a widely accepted formal foundation for service composition. It appears that Petri nets, process algebras, and state machines are suitable for capturing at least certain aspects of service composition. Ultimately, however, for a given formalism to be adopted in this area, it is necessary that its benefits are tangible (for example, availability of analysis and simulation tools) and that full mappings between this formalism and concrete modeling and description languages are provided.

Platforms for Composite Service Execution

The previous section explained the structure of an execution environment for composite services. In this section, we review existing implementations that serve as execution environments. We first provide an overview of some research prototypes before looking more closely at the implementations provided by major vendors: IBM, BEA, and Microsoft.

Research Prototypes

CMI (Collaboration Management Infrastructure) (Schuster et al., 2000) provides an architecture for interenterprise services. It uses state machines to describe the behavior of composite services. The concept of placeholder is used to enable the dynamic selection of services. A placeholder is a set of services (identified at runtime) and a method for selecting a service given a set of parameters.

eFlow (Casati & Shan, 2001) is a platform that supports the specification, enactment, and management of composite services. eFlow uses graph-based model in which the nodes denote invocations to service operations, and the edges denote control-flow dependencies. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The definition of a service node contains a search recipe represented in a query language. When a service node is invoked, a search recipe is executed to select a reference to a specific service. Once a service is selected by the search recipe, the eFlow execution engine is responsible for performing the dynamic binding using metadata that it stores in the service repository.

CrossFlow (Grefen et al., 2000) features the concept of contracts for services cooperation. When a partner wants to publish a collaboration, it uses its contract manager to send a contract template to matchmaking engine. When a consumer wants to outsource a service, it uses a contract template to search for relevant services. Based on the specifications in the contract, a service enactment structure is set up.

SELF-SERV (compoSing wEb accessibLe inFormation and buSiness services) (Benatallah, Dumas, Sheng & Ngu, 2002) specifies composite services using statecharts. Furthermore, SELF-SERV proposes a peer-to-peer model for orchestrating a composite service execution in which the control and data-flow dependencies encoded in a composite service definition are enforced through software components located in the sites of the providers participating in a composition. SELF-SERV refines the concepts of search recipe and placeholder introduced by eFlow and CMI by proposing the concept of community. A community is an abstract definition of a service capability with a set of policies for (i) managing membership in the community and (ii) selecting at runtime the service that will execute a given service invocation on behalf of the community. Policies for runtime selection of services are formulated using multiattribute value functions. A community is also responsible for performing the dynamic binding of the selected Web service, thereby acting as a dynamic service selector.

DySCo (Piccinelli, Finkelstein & Lane Williams, 2003) is another service-oriented workflow infrastructure, which supports the definition and enactment of dynamic service interactions. DySCo adopts a traditional workflow approach, except with respect to the definition of a task. Instead of corresponding to an activity involving a number of resources, a task in DySCo corresponds to an interaction step between services. In addition, DySCo supports the dynamic reconfiguration of service interactions by allowing a task to be decomposed at runtime into a more complicated structure. For example, a document mailing task in a service-based workflow can be decomposed into two tasks: a document printing task and a document posting task, which can then be assigned to different providers.

Commercial Tools

Typically, a tool claiming to support the Web services stack would minimally provide an API in one or more programming languages (for example, Java) for generating and/or processing SOAP messages. Some tools would go further by supporting tasks such as: (i) generating WSDL descriptions from modules, packages, or classes (for example, from a Java class file); (ii) editing WSDL descriptions through a graphical interface; and/or (iii) extracting information contained in WSDL files in order to dynamically generate stubs and skeletons that provide transparent communication between Web service requesters and providers.

Most tools also provide support for UDDI (both for setting up a registry and for connecting to an existing registry). Few tools currently support composite service description languages, and when they do, they typically only support a subset of these languages.

- **IBM WebSphere:** WebSphere is a family of IBM products for enabling B2B interactions. The application server is the cornerstone of WebSphere. It aims at providing database and backend integration as well as security and performance capability (for example, workload management). The WebSphere application server Advanced Edition adds support for J2EE and CORBA. The advanced edition integrates support for key Web service standards such as SOAP, UDDI, and WSDL. Additionally, it provides distributed transaction support for major database systems. Other products make up the WebSphere platform. These include WebSphere Business Components, WebSphere Commerce, and WebSphere MQ Family. The WebSphere Business Components provides prebuilt and tested components. WebSphere Commerce provides mechanisms for building B2B sites. WebSphere MQ Family is a family of message-oriented middleware products.
- **BEA WebLogic Integrator:** BEA WebLogic Integrator is one of the cornerstones of the BEA WebLogic e-Business Platform. It is built on top of a J2EE compliant application server and J2EE connector architecture and supports current Web service standards such as SOAP, UDDI, and WSDL. It is composed of four major modules:
  - The Application Server, which provides the infrastructure and functionalities for developing and deploying multitier distributed applications as EJB components.
  - The Application Integration Server, which leverages the J2EE connector architecture to simplify integration with existing enterprise applications, such as SAP R/3 and PeopleSoft.
  - The Business Process Management System, which provides a design tool and execution engine for business processes in BPEL4WS.
  - The B2B integration manages interactions with external business processes.

- **Microsoft Web Services Support:** Support for Web services is one of the key aspects of the .Net product series. In particular, ASP.Net provides a programming model for exposing applications as Web services. Briefly, the skeleton of a Web service is encoded as an ASMX file (proprietary Microsoft format), which can be interpreted by the Internet Information Server (IIS) in order to process incoming SOAP calls for the service and generate SOAP responses and faults. A WSDL description and a test page are also automatically generated from the ASMX file. Another Microsoft product, which provides support for Web services, is BizTalk: a middleware platform for Enterprise Application and B2B Integration. Applications in BizTalk are integrated based on an XML message-oriented paradigm. Part of the BizTalk suite is the BizTalk Orchestration Engine, which implements XLANG, a precursor of BPEL4WS. Developers can define processes using a graphical interface and export them as XLANG descriptions, which are then fed into the runtime engine.

Trends Relevant to (Web) Service Composition

While much of the work to date has focused on standards for announcing, discovering, and invoking Web services, there are other significant developments happening in Web services. In this section, we overview some of the developments related to conversation-driven composition, semantic Web services, and wireless Web services (also known as M-services), focusing on those aspects relevant to service composition.

Conversation-Driven Composition

A conversation is a consistent exchange of messages between participants involved in joint operations. A conversation succeeds when what was expected from that conversation in terms of outcome has been achieved. Further, a conversation fails when the conversation faced difficulties (for example, communication-medium disconnected) or did not achieve what was expected.

The use of conversations helps in defining composite services at runtime instead of design time. When a Web service is being executed, it has at the same time to initiate conversations with the Web services that are due for execution. The purpose of these conversations is twofold (Maamar, Benatallah & Mansoor, 2003): invite the Web services to join the composition process and ensure that the Web services are ready for execution in case they accept the invitation. Furthermore, conversations between Web services allow addressing of the composability problem. Medjahed, Rezgui, Bouguettaya, and Ouzzani (2003) note that an issue when defining a composite service is to check if the Web services can actually work together at the information level. Mapping operations of the parameters exchanged between Web services may be required. Ensuring the

composability of Web services can be completed using ontologies and conversations. Web services engage in conversations to agree on which ontology to use, what/how/when to exchange, and what to expect from an exchange.

The Web Services Conversation Language is an initiative on the integration of conversations into Web services. This language describes the structure of documents that a Web service is supposed to receive and produce and the order in which the exchange of these documents will occur. The conversation component to embed a Web service is mainly a means for describing the operations that a Web service supports (for example, clients have to log in first before they can check the catalogue).

Ardissono, Goy, and Petrone (2003) observed that current Web services communication standards support simple interactions and are mostly structured as question-answer pairs. These limitations hinder the possibility of expressing complex situations that require more that two turns of interactions (for example, propose/counter-propose/accept-reject). In addition, Ardissono et al. (2003) worked on a conversational model that aims at supporting complex interactions between clients and Web services, where several messages are exchanged before a Web service is completed.

It is stated that the full capacity of Web services as an integration platform will be reached only when applications and business processes integrate their complex interactions by using a standard process integration model such as BPEL4WS. While the orchestration of Web services is a core component to any Web services integration effort, the use of conversations gives more "freedom" to Web services to decide if they will take part in this orchestration. Conversations are more than just combining components; they promote the autonomy of components that act and react according to their environment (Hanson, Nandi & Levine, 2002).

Semantic Web Services

Another major trend is the integration of semantics into Web services. Heflin and Huhns (2003) argue that the goal driving the semantic Web is to automate Web-document processing. The semantic Web aims at improving the technology that organizes, searches, integrates, and evolves Web-accessible resources (for example, documents, data). This requires the use of rich and machine-understandable abstractions to represent the resource semantics.

One of the core components to the widespread acceptance of the semantic Web is the development of ontologies that specify standard terms and machine-readable definitions. Although there is no consensus yet on what an ontology is, most researchers in the field of knowledge representation consider a taxonomy of terms and the mechanisms for expressing the terms and their relationships. Samples of markup language for publishing and sharing ontologies on the 3W include RDF (Resource Description Framework), DAML+ OIL (DARPA Agent Markup Language + Ontology Inference Layer), and OWL (Web Ontology Language) (W3C World Wide Web Consortium, 2001).

By combining efforts of Web services and semantic Web communities, it is expected that new foundations and mechanisms for enabling automated discovery, access, combination, and management for the benefit of semantic Web services will be developed.

Paolucci and Sycara (2003) note that the semantic Web provides tools for explicit markup of Web content, whereas Web services could create a network of programs (that is, software agents) that produce and consume information, enabling automated business interactions. There exist various initiatives in the field of semantic Web services such as DAML-S (DARPA Agent Markup Language for Services) (DAML-S Consortium, 2004), WSMF (Web Services Modeling Framework) (Fensel & Bussler, 2002), and METEOR-S (Managing End-To-End OpeRations-Semantic Web Services and Processes) (Sivashanmugam, Verma, Sheth & Miller, 2003).

Wireless Web Services

Besides the Web expansion, a development occurring in the field of wireless and mobile technologies is witnessed (Wieland, 2003). Telecom companies are offering new services and opportunities to customers over mobile devices. The next stage (if we are not already in it), is to allow users to remotely enact Web services from mobile devices (Maamar & Mansoor, 2003).

While Web services provisioning is an active area of research and development (Benatallah & Casati, 2002), little has been done to date regarding their provisioning in wireless environments. This is due to different obstacles including throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections. In addition, businesses that are eager to engage in wireless Web services activities are facing technical, legal, and organizational challenges. To optimize Web services provisioning in wireless environments, important issues need to be tackled first:

Context-sensitive Web services selection: In addition to traditional criteria such as monetary cost and execution time, the selection of services should consider, on the one hand, the location of requesters and, on the other hand, the capabilities of the computing resources on which these services will be deployed (for example, processing capacity, bandwidth). This calls for context-aware service selection policies that enable a system to adapt itself to computing and user requirements.

Handling disconnections during Web services execution: In a wireless environment, disconnections are frequent. It is noted that to cope with disconnection issues during a service delivery, software agent-based service composition middleware architectures are deemed appropriate as proposed in Maamar, Sheng, and Benatallah (2004).

Conclusion

Web services promise to revolutionize the way in which applications interact over the Web. However, the underlying technology is still in a relatively early stage of development and adoption. While the core standards such as XML, SOAP, and WSDL are relatively stable and are supported in various ways by a number of tools, standardization efforts in key areas such as security, reliability, policy description, and composition are

still underway, and the tools supporting these emerging standards are still evolving. In addition (or perhaps as a result of this), relatively few production-level Web services have been deployed and are being used in practice. To some extent, these difficulties can be explained by the fact that businesses have spent considerable resources in the last few years to expose their functionality as interactive Web applications. As a result, they are reluctant to invest more to move this functionality into Web services until the benefits of this move are clear. It will probably take another two years before the technology reaches the level of maturity necessary to trigger a widespread adoption. In the meantime, it is important that middleware platform developers integrate the numerous facets of Web services into their products (for example, facilitating the use of message-oriented middleware for Web service development), while researchers advance the state of the art in challenging issues such as Web service delivery in mobile environments, QoS-driven selection of services, and manipulation of semantic-level service descriptions.

References

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2003). *Web services: Concepts, architectures and applications*. Berlin: Springer-Verlag.

Apache Web Services Project. (2003). Web services invocation framework (WSIF). Retrieved August 6, 2004: *http://ws.apache.org/wsif/*

Ardissono, L., Goy, A., & Petrone, G. (2003, July 14-18). *Enabling conversations with web services*. Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Melbourne, Australia.

BEA Systems, Microsoft, IBM & SAP. (2003). Business process execution language for Web services (BPEL4WS). Retrieved August 6, 2004: *ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf*

Benatallah, B., & Casati, F. (2002). Introduction to special issue on Web services. *Distributed and Parallel Databases: An International Journal, 12*(2-3).

Benatallah, B., Dumas, M., Sheng, Q., & Ngu, A. (2002, February 26-March 1). *Declarative composition and peer-to-peer provisioning of dynamic Web services*. Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE), San Jose, CA.

Benatallah, B., Sheng, Q., & Dumas, M. (2003). The SELF-SERV environment for Web services composition. *IEEE Internet Computing, 7*(1), 40-48.

Bultan, T., Fu, X., Hull, R., & Su, J. (2003, May 20-24). *Conversation specification: A new approach to design and analysis of e-service composition*. Proceedings of the 12th International Conference on the World Wide Web (WWW'03) (pp. 403-410), Budapest, Hungary.

Bussler, C. (2003). *B2B integration: Concepts and architecture*. Berlin: Springer-Verlag.

Cardelli, L., & Davies, R. (1999). Service combinators for Web computing. *IEEE Transactions on Software Engineering*, *25*(3), 309-316.

Casati, F., & Shan, M.-C. (2001). Dynamic and adaptive composition of e-services. *Information Systems*, *26*(3), 143-162.

Chen, Q., & Hsu, M. (2002, October 28-November 1). *CPM revisited – An architecture comparison.* Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE (pp. 72-90), Irvine, CA.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, *6*(2), 86-93.

DAML-S Consortium. (2004). DAML services. Retrieved August 6, 2004: *http://www.daml.org/services*

Fauvet, M.-C., Dumas, M., & Benatallah, B. (2002, October 28-November 1). *Collecting and querying distributed traces of composite service executions.* Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE (pp. 373-390), Irvine, CA.

Fensel, D., & Bussler, C. (2002). The Web services modeling framework WSMF. *Electronic Commerce Research and Applications*, *1*(2), 113-137.

Grefen, P., Aberer, K., Hoffner, Y., & Ludwig, H. (2000). CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science & Engineering*, *15*(5), 277-290.

Hanson, J. E., Nandi, P., & Levine, D. W. (2002, June 24-27). *Conversation-enabled Web services for agents and e-business.* Proceedings of the International Conference on Internet Computing (IC), Las Vegas.

Heflin, J., & Huhns, M. (2003). The Zen of the Web. *IEEE Internet Computing*, *7*(5).

Maamar, Z., Benatallah, B., & Mansoor, W. (2003, May 20-24). *Service chart diagrams: Description and application.* Proceedings of the 12th International Conference on the World Wide Web (WWW'03), Budapest, Hungary.

Maamar, Z., & Mansoor, W. (2003). Design and development of a software agent-based and mobile service-oriented environment. *e-Service Journal*, *2*(3).

Maamar, Z., Sheng, Q.Z., & Benatallah, B. (2004). On composite web services provisioning in an environment of fixed and mobile computing resources. *Information Technology and Management Journal, 5*(3).

Mecella, M., Parisi-Presicce, F., & Pernici, B. (2002, August 23-24). *Modeling e-service orchestration through Petri nets.* Proceedings of the 3rd International Workshop on Technologies for E-Services (TES) (pp. 38-47), Hong Kong.

Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A., & Elmagarmid, A. (2003). Business-to-business interactions: Issues and enabling technologies. *The VLDB Journal, 12*(1), 59-85.

Medjahed, B., Rezgui, A., Bouguettaya, A., & Ouzzani, M. (2003). Infrastructure for e-government Web services. *IEEE Internet Computing*, *7*(1).

Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A., & Weikum, G. (1998). From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, *10*(2).

Narayanan, S., & McIlraith, S. (2002, May 7-11). *Simulation, verification and automated composition of Web services*. Proceedings of the 11th International Conference on the World Wide Web (pp. 77-88), Honolulu.

Paolucci, M., & Sycara, K. (2003). Autonomous semantic Web services. *IEEE Internet Computing*, *7*(5).

Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, *36*(8), 46-52.

Piccinelli, G., Finkelstein, A., & Lane Williams, S. (2003, September 1-6). *Service-oriented workflow: The DySCo framework.* Proceedings of the 29th EUROMICRO Conference (pp. 291-297), Belek-Antalya, Turkey.

RosettaNet (2004). RosettaNet home page. Retrieved August 6, 2004: *http://www.rosettanet.org*

Schuster, H., Georgakopoulos, D., Cichocki, A., & Baker, D. (2000, June 5-9). *Modeling and composing service-based and reference process-based multi-enterprise processes.* Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE) (pp. 247–263), Stockholm, Sweden.

Sivashanmugam, K., Verma, K., Sheth, A., & Miller, J. (2003, October 20-23). *Adding semantics to web services standards*. Proceedings of the 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL.

UN/CEFACT, & OASIS (2001a). Collaboration-protocol profile and agreement specification. Retrieved August 6, 2004: *http://www.ebxml.org/specs/ebCCP.pdf*

UN/CEFACT, & OASIS (2001b). ebXML business process specification schema. Retrieved August 6, 3004: *http://www.ebxml.org/specs/ebBPSS.pdf*

van der Aalst, W. (2003). Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, *18*(1).

van der Aalst, W., & van Hee, K. (2002). *Workflow management: Models, methods, and systems*. Cambridge, MA: MIT Press.

W3C World Wide Web Consortium. (2002). Web services choreography working group. Retrieved August 6, 2004: *http://www.w3.org/2002/ws/chor*

W3C World Wide Web Consortium. (2001). Semantic Web activity. Retrieved August 6, 2004: *http://www.w3.org/2001/sw*

Wieland, K. (2003). The long road to 3G. *International Telecommunications Magazine*, *37*(2).

Wohed, P., van der Aalst, W., Dumas, M., & ter Hofstede, A. (2003, October 13-16). *Analysis of Web services composition languages: The case of BPEL4WS.* Proceedings of the 22nd International Conference on Conceptual Modeling (ER). Chicago.

Section II

Service-Oriented Architecture Design and Development

Chapter IV

UniFrame: A Unified Framework for Developing Service-Oriented, Component-Based Distributed Software Systems

Andrew M. Olson
Indiana University Purdue University, USA

Rajeev R. Raje
Indiana University Purdue University, USA

Barrett R. Bryant
University of Alabama at Birmingham, USA

Carol C. Burt
University of Alabama at Birmingham, USA

Mikhail Auguston
Naval Postgraduate School, USA

Abstract

This chapter introduces the UniFrame approach to creating high quality computing systems from heterogeneous components distributed over a network. It describes how this approach employs a unifying framework for specifying such systems to unite the concepts of service-oriented architectures, a component-based software engineering methodology and a mechanism for automatically finding components on a network in order to assemble a specified system. UniFrame employs a formal specification language to define the components and serve as a basis for generating glue/wrapper code that connects heterogeneous components. It also provides a high level language for the

system developer to use for inserting code in a created system to validate it empirically and estimate the quality of service it supports. The chapter demonstrates how a comprehensive approach, which involves the practicing community as well as technical experts, can lead to solutions of many of the difficulties inherent in constructing distributed computing systems.

Introduction

The architecture of a computing system family can be represented by a business model comprising a set of standard, platform independent models residing in a service layer, each of which is related to a platform specific model that corresponds to one or more specific realizations of the service. A system is realized by assembling the realizations according to the specified architecture. This Service-Oriented Architecture offers many advantages, such as flexibility, in constructing and modifying a computing system. Because business requirements can change rapidly, both the services making up a business model and their platform specific realizations may need to change rapidly in response. With an agile mechanism to trace out an appropriate architecture, the development engineer can react quickly by building a modified realization of the system. Nevertheless, there are many practical issues that make effecting this process difficult. For example, an environment in which this approach has greatest appeal is typically distributed and heterogeneous. This makes the mapping of a system's platform independent model to a platform specific model (Object Management Group, 2002) quite complex and subject to variation.

This chapter describes the basic principles of the UniFrame Project, which defines a process, based on Service-Oriented Architecture, for rapidly constructing a distributed computing system that confronts many of these inherent difficulties. UniFrame's basic objective is to create a unified framework to facilitate the interoperation of heterogeneous distributed components as well as the construction of high quality computing systems based on them. UniFrame combines the principles of distributed, component-based computing, Model-Driven Architecture, service and quality of service guarantees, and generative techniques.

Though better than handcrafting distributed computing systems, developing them by composing existing components still poses many challenges. A comprehensive treatment of these and the corresponding solutions that UniFrame proposes exceeds the scope of this chapter, so it sketches the features of UniFrame that are most related to the book's service-oriented engineering theme along with references to further reading.

Background

Despite the achievements in software engineering, development of large-scale, decentralized systems still poses major issues. Recent experience has demonstrated that the

principles of distributed, component-based engineering are effective in dealing with them. Weck (1997), Lumpe, Schneider, Nierstrasz, and Achermann (1997), and the works of Batory et al., for example, Batory and Geraci (1997), concern the composition of components. The approach of Griss (2001) to developing software product lines is similar to UniFrame's, except that UniFrame avoids descending to code-fragment-sized components. Brown (1999) surveys component-based system development, whereas Heineman and Councill (2001) and Szyperski, Gruntz, and Murer (2002) provide extensive discussions of different aspects.

Heineman and Councill (2001) provide a general definition of a component model. Many different models for distributed, component-based computing have been proposed and implemented. Among these, J2EE™ (Java 2 Enterprise Edition) and its associated distributed computing model (Java-RMI), CORBA® (Common Object Request Broker Architecture), and .NET® have achieved the greatest acceptance. Typically, each prevalent model assumes the presence of homogeneous environments; that is, components created using a particular model assume that any other components present adhere to the same model. For example, the white paper on Java Remote Method Invocation (2003) describes RMI as an extension of Java's basic model to achieve distributed computation, assuming, thus, an environment consisting of components developed using Java and communicating with each other using method calls. Schmidt (2003) provides an overview of CORBA, which indicates that CORBA does provide a limited independence from the components' development language and deployment platform by specifying components with an interface definition language. This permits implementation in any languages for which mappings with the interface definition language exist. Again, an implicit assumption is that, typically, a CORBA component will communicate with another CORBA component. Microsoft's .NET is intended as a programming model for building XML™-based Web services and associated applications. It provides language independence with an interface language and a common language runtime (Microsoft .NET Framework, 2003). The implicit assumption of homogeneity still holds.

UniFrame

Current approaches for tackling heterogeneity are *ad hoc* in nature, requiring handcrafted software bridges, so have many drawbacks. It is difficult to make components of different models interoperate, and handcrafting is known to be error prone. Moreover, dependence on a single model meshes poorly with the grand notion of a component (or services) bazaar over a distributed infrastructure, as the success of such a bazaar requires local autonomy for deciding various policies, including the choice of the underlying model. Thus, there is a need for a framework, such as UniFrame, that will support seamless interoperation of heterogeneous, distributed components. UniFrame consists of:

- the creation of a standards-based meta-model for components and associated hierarchical setup for indicating the contracts and constraints of the components;

- an automatic generation of glue and wrappers for achieving interoperability;
- guidelines for specifying and verifying the quality of individual components;
- a mechanism for automatically discovering appropriate components on a network;
- a methodology for developing distributed, component-based systems with service-oriented architectures; and
- mechanisms for evaluating the quality of the resulting component assemblages.

UniFrame creates more general distributed systems than the point-to-point interactions of current Web services and also emphasizes determining the Quality of Service (QoS) during system assembly. For pragmatic reasons, UniFrame provides an iterative, incremental process for assembling a distributed computing system (DCS) from services available on the network that permit selecting among alternative components during system construction. In order to increase the assurance of a DCS, UniFrame employs automation, to the extent feasible, in the processes of locating and assembling components, and of component and system integration testing. The ICSE 6th Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction (Crnkovic, Schmidt, Stafford & Wallnau, 2003) focused on automated composition theories in constructing a DCS. Although automation is a goal of UniFrame, it presently focuses on the more practical, implementation aspects.

Unified Meta-Component Model (UMM)

Because future service-oriented systems will consist of independently developed components adhering to various models, a meta-model that abstracts the features of different models, enhances them and incorporates innovative concepts, is necessary in order to facilitate their creation. Raje (2000) and Raje, Auguston, Bryant, Olson, and Burt (2001) describe a central concept of UniFrame, the Unified Meta-component Model, that does this. It consists of three parts: (a) components, (b) service and its guarantees, and (c) infrastructure. These are not novel separately, but their structure, integration, and interactions form the UMM's distinguishing features. Components in the UMM have public interfaces and private implementations, which may be heterogeneous. Each interface comprises multiple levels. In addition to emphasizing a component's functional responsibilities (or the services it offers), the UMM requires component developers to advertise and guarantee a QoS rating for each component. The UMM's infrastructure supplies the environment necessary for developing, deploying, publishing, locating, assembling, and validating individual components and systems of components. The following subsections expand upon these concepts.

Component

The UMM defines a component as a sextuple consisting of the attributes (inherent, functional, nonfunctional, cooperative, auxiliary, deployment). This view of a component conforms to the definition of Szyperski, Gruntz, and Murer (2002). The inherent attributes contain the bookkeeping information about a component, such as the author, the version, and its validity period. The functional attributes of a component contain its interface, along with the necessary pre- and post-conditions, and component model of any associated implementation. They also indicate related details, such as algorithms used, underlying design patterns and technology, and known usages. The nonfunctional attributes represent the QoS parameters supported by the component, along with their values that the component developer guarantees in a specific deployment environment. These attributes may also indicate the effects of the deployment environment and usage patterns on the QoS values. The cooperative attributes describe how components actively collaborate, exchanging services. The auxiliary attributes exhibit other characteristics, such as mobility, various security features, and fault tolerance that the components may possess. A component needs deployment rules, specified in the deployment attributes so that it can be configured, initialized, and made available on a network.

Service

As described by Raje (2000), this part of the UMM consists of the computational tasks and guarantees that a component performs. To realize a DCS from a set of independently created components, the system integrator needs to reason from the service assurance of each component to obtain the assurance of the integrated DCS. Hence, a component must provide a predetermined level of assurance of both its functional and nonfunctional features. Various techniques, such as formal verification, have been proposed for reasoning about the functional assurance of a DCS. Therefore, the UMM assumes the use of an appropriate mechanism for functional assurance. The UniFrame research focuses on assuring the nonfunctional features of components and the integrated system because many existing application domains (multimedia, critical systems, and so forth) depend not only on correct functionality but also on how well it is achieved. UniFrame provides a mechanism for the component provider to specify the QoS parameters that are applicable to a provided component and determine the ranges that the component can guarantee.

Table 1 shows the UMM type specification of a component, *Validation Server*, for validating user accesses within the application domain of document management. In the advertised description of a corresponding implementation, the component provider would supply the actual values for various fields (such as N/A in Table 1). For example, the specification of a component that implements *Validation Server* would contain details, such as the URL where the component is deployed (id), the guaranteed values for the *throughput* and *end-to-end delay*, and the required deployment environment. The

Table 1. UMM type specification of a component

Abstract Component Type: *ValidationServer*
1. Component Name: *ValidationServer*
2. Domain Name: Document Management
3. System Name: DocumentManager
4. Informal Description: Provide the user validation service.
5. Computational Attributes:
5.1 Inherent Attributes:
5.1.1 id: N/A
5.1.2 Version: version 1.0
5.1.3 Author: N/A
5.1.4 Date: N/A
5.1.5 Validity: N/A
5.1.6 Atomicity: Yes
5.1.7 Registration: N/A
5.1.8 Model: N/A
5.2 Functional Attributes:
5.2.1 Function description: Act as validation server for users in the system.
5.2.2 Algorithm: N/A
5.2.3 Complexity: N/A
5.2.4 Syntactic Contract
5.2.4.1 Provided Interface: *IValidation*
5.2.4.2 Required Interface: NONE
5.2.5 Technology: N/A
5.2.6 Expected Resources: N/A
5.2.7 Design Patterns: NONE
5.2.8 Known Usage: Validation of user access
5.2.9 Alias: NONE
6. Cooperation Attributes:
6.1 Preprocessing Collaborators: *Users'Terminal*
6.2 Postprocessing Collaborators: NONE
7. Auxiliary Attributes:
7.1 Mobility: No
7.2 Security: *L0*
7.3 Fault tolerance: *L0*
8. Quality of Service Attributes
8.1 QoS Metrics: *throughput*, *end-to-end delay*
8.2 QoS Level: N/A
8.3 Cost: N/A
8.4 Quality Level: N/A
8.5 Effect of Environment: N/A
8.6 Effect of Usage Pattern: N/A
9. Deployment Attributes: N/A

specification associated with each implemented component is published when it is deployed on the network. The UMM specification of a component enhances the concept of a multilevel contract for components proposed by Beugnard, Jezequel, Plouzeau, and Watkins (1999) because it includes other details, such as bookkeeping, collaborative, algorithmic and technological information, and possible levels of service with associated costs and effects of different environmental factors on the QoS parameters.

Infrastructure

UniFrame assumes the presence of a publicly accepted knowledgebase that contains information, such as the component types needed for a specific application domain, the interconnections and constraints that make up the design specification of each component system in a domain, and rules for QoS calculations. Experts, such as standards organizations' task forces, create the UMM specifications for the components of each application domain of the knowledgebase. The UMM specifications of the component types are publicly distributed so that component developers can supply implementations that adhere to them.

UniFrame's Infrastructure consists of the System Generation Process, Resource Discovery Service (URDS), and Glue and Wrapper Generator. The first employs the knowledgebase to carry out the steps in creating a component system. It invokes the URDS to locate the components in the network the system requires and validates the product using an iterative process. The URDS provides mechanisms for components to publish their UMM specifications and for hosting the services on distributed machines, receives appropriate queries for locating the deployed services, and performs the selection of necessary components based upon specified criteria. It invokes the Glue and Wrapper Generator, which accommodates the heterogeneity across components, incorporates the mechanisms necessary to measure the QoS, and configures the selected services. Subsequent sections will provide more details about these.

Service-Oriented Architecture

In order to provide flexible, efficient support to the process of creating a DCS, UniFrame organizes its knowledgebase according to the concepts of Model-Driven Architecture proposed by the Object Management Group (2002) and Business Line Architecture proposed by the Enterprise Architecture SIG (2003a). UniFrame's UMM provides an underlying framework for this organization. The domain elements in the top tier of the architecture correspond to different business contexts, or lines. A context consists of a class of related business practice domains (such as, retail grocery, retail hardware, construction supply, wholesaler), which are located in the next tier down. Conceptually, elements on one level can share an element on another (health care and construction can share inventory), which differs in how it performs similar operations in different contexts (that is, the element comprises a set of variants). The various, hierarchically organized elements that contribute detail to the definition of a business context constitute its Business Reference Model, discussed in Succeeding with Component-Based Architecture by the Enterprise Architecture SIG (2003b). This takes the form of a tree, whose root represents the context in the architecture under consideration. Business domain experts perform requirements analysis and model the business contexts for which it is desired to construct DCSs. The Business Reference Models they derive and place in the knowledgebase define the space of problems UniFrame can solve.

For each Business Reference Model, software engineers construct design models in various ways to implement DCSs that satisfy its requirements. A design model is expressed, frequently in Unified Modeling Language (UML®) (Rumbaugh, Jacobson & Booch, 1999), in terms of tiered layers of components, each component offering a defined set of services. Several Business Reference Models can share components. A component in one tier can be composed (or use of the services) of components on a lower tier. Thus, a component has two definition forms in the knowledgebase:

- a specification of its abstract properties as a type, as in Table 1, or
- a design specification, following UMM standards, which directly references the components and refined design specifications that it uses.

The former is called an *abstract component*, which the UniFrame System Generation Process considers to be available with no construction necessary. The second form is called a *compound component*. The process will attempt to construct it from its design. A design specification that defines a realization of a Business Reference Model forms a Service Reference Model for it. It provides a vehicle for realizing the Model-Driven Architecture's mapping from a platform-independent model to a platform-specific model. The Service Reference Models also form part of UniFrame's knowledgebase.

In order to construct DCS solutions for a significant space of problems, the knowledgebase must contain matching (Business Reference Model, Service Reference Model) pairs for each problem variation anticipated. These can be organized efficiently by structuring related Business Reference Models in feature models according to the optional features that they exhibit and related Service Reference Models according to variation point archetypes that show which design variants are available. The experts create a domain-specific language based on the distinguishing features and variation points in the models. Then, users of the System Generation Process employ the language to specify their requirements. The following example illustrates the knowledgebase's organization.

Case Study

Suppose domain experts want to create a knowledgebase that includes the business context consisting of users who manage documents. The users' contact with the supporting system is via the use case *Manage Documents*, which includes *Validate User*. The use cases *Create Document, Delete Document*, *List Documents*, *Store Document,* and *Get Document* all extend *Manage Documents*. The last in this list includes *Lock Document*, whereas the others include *Unlock Document*. From the requirements these express, the domain experts identify three subsystems comprising the system: one for user validation, one for managing the documents themselves, and one for user interaction. The experts write a domain model for this system containing these three subsystems.

Suppose the experts decide the users may want to choose between two types of document manager systems: a standard document manager and a deluxe one that provides extended persistence support. They represent these options in a simplified feature diagram for the document manager, as shown in Figure 1. Clear small circles indicate optional features, whereas an arc indicates an exclusive OR choice. In more general feature diagrams (Griss, 2001), options of a node can be chosen as any combination of elements of a subset of the node's children. A feature diagram carries no information about how its alternatives might be associated with elements in the domain model of their parent node. It is an efficient mechanism for representing alternatives; the domain models are essential for representing the associations among elements in the models and the constraints on them. The domain model for the standard document manager consists of only one domain element, *Document Server*. The domain model for the deluxe document manager consists of two domain elements, *Deluxe Document Server* and its associated *Document Database* for persistence. Because there are just two alternatives in the feature diagram, there are just two Business Reference Models in this example. More generally, there will be as many as there are combinations permitted by the various feature diagrams present in the knowledgebase.

Software engineers experienced in the domain of the business context (document management here) develop design models for these two Business Reference Models. They create a service-oriented architecture of abstract components so that domain models map to component-based design models. Figure 2 shows the Service Reference Model, *Standard Document System*, for the Business Reference Model of the *Standard Document Manager* for this example. The Service Reference Model, *Deluxe Document System*, for the *Deluxe Document Manager* is identical, with the addition of a *Database* component associated with the *Document Server*, where the cardinality allows an arbitrary, positive number of *Database* units to be present. The Service Reference Models include the details defining the associations among the components. These might be views consisting of UML collaboration diagrams. This information is used to determine the entries in the UMM abstract component specifications and the interrelations of the components' interfaces. The specification for the abstract component, *Validation Server*, appeared in Table 1.

Suppose that the software engineers decide that two implementations of the standard document manager are possible, one in which the components adhere to .NET and the

Figure 1. Feature diagram for the document management system

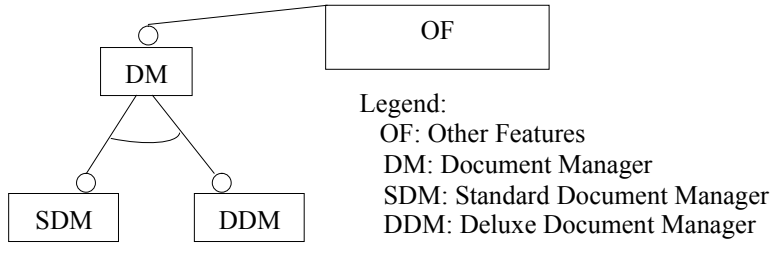
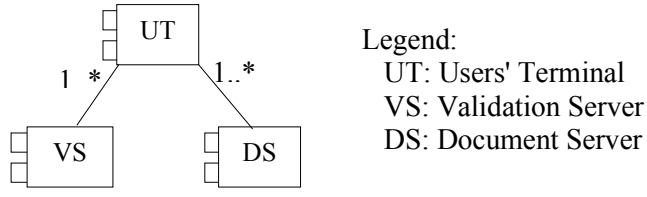


Figure 2. Service reference model for the standard document system



other to CORBA. They indicate this choice by a design model, labeled *Standard Document System*, augmented by variation point information that specifies the choice of one of these two technologies for the associations in Figure 2, such as in OCL (Warmer & Kleppe, 2003), as shown:

context Standard Document System
inv: technology = '.NET' or technology = 'CORBA'

Because the system consists of more than two components, the engineers have other combinations possible. For example, the *Users' Terminal/Validation Server* association may be in .NET technology, and the *Document Server* may be in CORBA technology, implying the need for an appropriate bridge.

UniFrame System Generation Process

The essential steps in UniFrame's process of constructing a DCS to solve a problem appear in Table 2. Once the UniFrame knowledgebase is available, a system developer can pose a statement of requirements for a DCS that solves a problem within its application domain. This analysis task forms step (1) in Table 2. For the case study in the previous section, the statement of requirements might be:

Create a Document Management System having a Standard Document Manager.

In step (2), the term *Document Management System* of the example requirements statement identifies the business context, so the stated problem lies within the domain the knowledgebase represents. The corresponding system model shows there are two alternatives for the *Document Manager,* which the feature model displays in Figure 1. The qualifying requirement, *Standard,* resolves this ambiguity, which completes step (2). The resulting Business Reference Model maps directly in the knowledgebase to the two alternative platform-specific Service Reference Models for the entire system shown in

Table 2. Steps in the UniFrame System Generation Process

Steps	Activities
1	State the requirements the DCS must satisfy in the knowledgebase's terminology.
2	Identify a Business Reference Model that represents these.
3	Identify each Service Reference Model specifying a system of abstract components that satisfies the Business Reference Model.
4	Obtain concrete implementations of the abstract components.
5	Assemble the concrete components into a DCS according to each Service Reference Model, so that it meets the specified requirements.
6	Test the DCS against the requirements and exit if satisfactory; otherwise, return to step (1) to modify the requirements.

Figure 2, in which the components are either all .NET or all CORBA. This completes step (3).

Continuing to step (4), the System Generation Process collects the UMM type specifications of all the abstract components involved in each of the two Service Reference Models and sends them in a query to the UniFrame Resource Discovery Service. This searches the network for implemented components whose UMM descriptions satisfy the type specifications.

Step (5) employs the design information in a Service Reference Model to construct a DCS with the components found. If the appropriate implementations are available on the network, the request for a *Standard Document Manager* in the example will yield two DCSs, one with .NET technology and one with CORBA technology. If no .NET implementation of a *Validation Server* is found, then only the CORBA DCS will be constructed.

Typically, a developer understands the requirements poorly at the initiation of the System Generation Process. Therefore, it is imperative to evaluate empirically the consistency of the characteristics of a generated DCS with the perceived requirements and make modifications as necessary. This motivates having step (6) in Table 2. Such iterative development provides a mechanism for the developer to validate the outcome of the process and determine empirically the ranges within which its QoS attributes vary. This helps to assure a higher quality product. The process allows two levels of testing. The simplest is black box (or acceptance) testing of the DCS based on only the stated requirements. The developer supplies a test harness and plan for this. The other is white box (or integration) testing, again based on the developer's test plan. In this case, the design of the DCS serves as a guide for inserting instrumentation code between the components in the DCS. At runtime, this code reports the behavior of the DCS, giving the developer a view into its internal operation. The section on the measurement of QoS discusses a mechanism for inserting this instrumentation easily.

In case there are several Business or Service Reference Models in the knowledgebase that satisfy the developer's requirements if step (2) or (3) of the process provides

feedback, allowing the developer to introduce requirements incrementally so as to reduce these alternatives, then the process becomes an efficient way to construct the needed type of DCS. Thus, the System Generation Process supports the iterative, incremental development paradigm that modern software engineering practices have found productive.

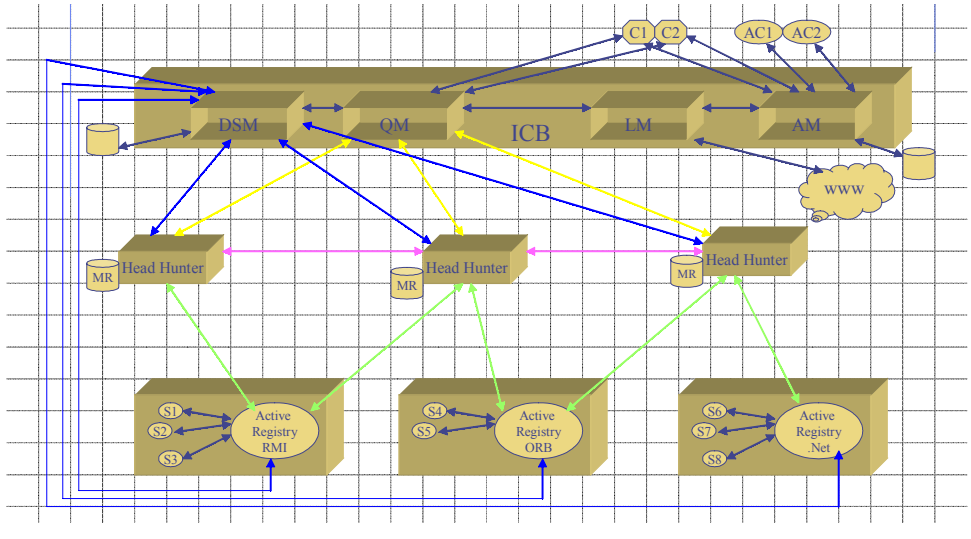
UniFrame Resource Discovery Service (URDS)

Once components and their UMM descriptions have been deployed on the network, they are ready for discovery in the UniFrame System Generation Process. The URDS executes this process. Siram et al. (2002) discuss its architecture, shown in Figure 3.

The URDS architecture comprises: HeadHunters (HHs), Internet Component Broker (ICB), Meta-repositories (MRs), and components.

Components are implemented according to some component model, as described earlier, and registered with the model's binding service. For example, the Java-RMI components are registered with the Naming service provided by the Java-RMI framework. An

Figure 3. UniFrame Resource Discovery System (URDS)



advantage of this is that it does not burden the component providers because, to deploy their implementations, they must register them anyway. The HHs have the sole responsibility of performing matchmaking operations between registered components and requested specifications. Each HH has an MR, which serves as a local store. An HH is constantly discovering newly implemented components and storing their UMM specifications in its MR. Anytime an HH receives a query for a component type, it first searches its MR. If it finds a match, it returns the corresponding component as a result. If not, it propagates the query to other HHs in the system.

The ICB is analogous to the object request broker (ORB) in other architectures. Unlike the ORB, which only allows interoperation between components having heterogeneous implementations, the Internet component broker allows interoperation between components with different component models. As Figure 3 shows, the Internet component broker consists of domain security manager (DSM), query manager (QM), link manager (LM), and adapter manager (AM). The DSM is responsible for enforcing a security structure on the URDS. It authenticates the HHs and allows them to communicate with different binding mechanisms (registries). The QM interfaces with the System Generation Process. It receives a query consisting of a collection of UMM component types, passes it to the HHs, and returns the results. The LM allows a federation of URDSs to be created in order to increase the component search space. The AM locates adapter components, such as bridges that allow interoperation of different component models, and passes them to the Glue and Wrapper Generator.

A prototype of URDS has been implemented using the Java-RMI and .NET technologies. Many experiments have been performed to measure its performance (Siram et al., 2002). These demonstrate that URDS scales upward, but the details extend beyond this chapter's scope.

Industry and academia have proposed and implemented many distributed resource discovery and directory services. Examples that Siram et al. (2002) describe include WAIS, Archie, Gopher, UDDI, CORBA Trader, LDAP, Jini, SLP, Ninf, and NetSolve. Each has its own characteristics and exhibits some similarity with URDS. The distinguishing features of URDS are its treatment of heterogeneity and its purpose to support creating heterogeneous integrated systems, not just to discover services.

UniFrame Quality of Service Framework (UQoS)

Components offer services and indicate and guarantee the quality of their services. Therefore, it is necessary to facilitate the publication, selection, measurement, and validation of component and DCS QoS values. The UniFrame Quality of Service Framework, described by Brahnmath (2002); Sun (2003); and Raje, Bryant, Olson, Auguston, and Burt (2002), provides necessary guidelines for the component developers and system integrators using UniFrame. The UQoS consists of three parts: QoS catalog,

composition/decomposition models for QoS parameters, and specification and measurement of QoS. The reader is referred to the references above for the first two because the details are extensive.

To prepare the UMM description of a component to be publicized, the component developer must measure empirically the QoS parameters in the corresponding UMM type specification. The QoS catalog provides model definitions and formulas to assist in this. Some parameters are static in nature (like reliability), while some are dynamic (like end-to-end delay). If the parameter is static and characterizes a system of components, then its value can be determined from the components' parameter values. Otherwise, its value must be determined empirically.

Evaluation of QoS Parameters

UniFrame uses the principles of event grammars for measuring parameters empirically. Event grammar, as described by Auguston (1995), forms the basis for system behavior models. An event represents any detectable action during execution, such as a statement execution, expression evaluation, procedure call, and receiving a message. It has a beginning, end, and duration (a time interval corresponding to the action of interest). Actions (or events) evolve in time, and system behavior represents the temporal relationship among actions. This implies a partial ordering relation for events, as Lamport (1978) discussed.

System execution can be modeled as a set of events (event trace) with two basic relations: partial ordering and inclusion. The event trace actually is a model of the system's temporal behavior. In order to specify meaningful system behavior properties, events must be enriched with attributes. An event may have a type and other attributes, such as duration, source code related to the event, associated state (that is, variable values at the event's beginning and end), and function name and returned value for function call events.

A special programming language, FORMAN, for computations over event traces greatly facilitates measuring parameters empirically. As described by Fritzson, Auguston, and Shahmehri (1994) and Auguston (1995), it is based on the notions of the functional paradigm, event patterns, and aggregate operations over events.

The execution model of a component (or a system of integrated components) is defined by an event grammar, which is a set of axioms that describes possible patterns of basic relations between events of different types in a program execution trace. It is not intended to be used for parsing actual event traces. If an event is compound, the grammar describes how it splits into other event sequences or sets. For example, the event *execute-assignment-statement* contains a sequence of events *evaluate-right-hand-part* and *execute-destination*.

The rule *A :: (B C)* establishes that, if an event *a* of the type *A* occurs in the trace of a program, it is necessary that events *b* and *c* of types *B* and *C*, also exist, such that the relations *b* IN *a*, *c* IN *a*, *b* PRECEDES *c* hold. For example, the event grammar describing

the semantics of an imperative programming language may contain the following rule (the names, such as *execute-program* and *ex-stmt* in the grammar denote event types):

*execute-program :: (ex-stmt *)*

This means that each event of the type *execute-program* contains an ordered (w.r.t. relation PRECEDES) sequence of zero or more events of the type *ex-stmt*. For the function call event, the event grammar may provide the following rule:

*func_call:: (param *) (ex-stmt *)*

This event may contain zero or more parameter evaluation events followed by statement executions.

Example of Evaluating Turn-Around Time

If the event type *component_call* corresponds to the whole component call event and *request* denotes the event for a single request (the time interval from the request's beginning to its completion), then the following FORMAN formula specifies the measurement of the turn-around time:

```
FOREACH a: session FROM execute_program
SAY ( 'Turn-around Time for a session is '
    SUM[ b: request FROM a APPLY b.duration]
                    / CARD[ request FROM a] )
```

Similar rules can be specified for any other dynamic QoS parameters or related computations. Thus, the principles of event traces provide a mechanism to validate empirically the QoS values for a component and for an integrated system of components.

Interoperability Using the Glue and Wrapper Generator

For interoperation of heterogeneous distributed components, it is necessary to construct glue and wrapper code to interconnect the components. Because a project objective is to achieve high quality systems, a goal is to automatically generate the glue/wrapper code. In order to achieve this, there should be formal rules for interconnecting

components from a specific application domain as well as integration of multiple technology domains, that is, component models. UniFrame uses the Two-Level Grammar (TLG, also called W-grammar) formal specification language (Bryant & Lee, 2002) to specify both types of rules. The TLG formalism is used to specify the components deployed under UniFrame and also the generative rules needed for system assembly. The output of the TLG will provide the desired target code (for example, glue and wrappers for components and necessary infrastructure for the distributed runtime architecture). The UMM formalization establishes the context for which the generative rules may be applied. Bryant, Auguston, Raje, Burt, and Olson (2002) provide further details about the glue/wrapper code generation rules, including a discussion of how the Quality of Service validation code is inserted into the glue code. The general principle is that for each QoS parameter to be dynamically verified, the glue code is instrumented according to the event grammar rules described earlier.

Future Trends

The concept of Business Reference Models "is meant to provide the foundation for common understanding of business processes across the Federal government in a service-oriented manner," enabling an agency to define an enterprise architecture as mandated by law (Enterprise Architecture SIG, 2003). A significant sector of industry is involved in establishing standards and guidelines on how to enable successful enterprise architecture. The component-based architecture of UniFrame's knowledgebase closely follows these guidelines, incorporating the concepts of Object Management Group's (2002) Model-Driven Architecture as an integral part. Consequently, UniFrame is working toward the realization of an operational framework for enterprise architecture and is a source of feedback into the activities necessary.

Many existing component models provide the necessary mechanisms for describing the functional aspects of components but not for the QoS aspects. Standards organizations have recently started to address this weakness. For example, in the fall of 2000, the OMG began issuing a number of Requests for Proposals for UML profiles for modeling QoS in several contexts. UniFrame is addressing some of these QoS issues and is making efforts (via presentations to different OMG task forces) to ensure that its research is aligned with emerging industry standards.

The creation of the Business Line and Service-Oriented knowledgebase will largely continue to be a human endeavor aided by CASE tools because humans determine what constitutes the problems they must solve. However, the System Generation Process could be accomplished mostly automatically for any problem in a given knowledgebase. The person who formulates the requirements for the DCS will need to do so in the knowledgebase's terminology. The degree to which this can be made to match the typical user's terminology remains a research area.

Huang (2003) implemented a prototype of the UniFrame System Generation Process with the UniFrame Resource Discovery Service. Because of the labor involved in constructing the knowledgebase, it was limited to a small banking case study. Experimental studies

proved efficient, user communication issues were easily managed, and QoS values were calculated. The automated creation of bridges and glue/wrapper code and using FORMAN to insert the code into them for the QoS computations remain to be incorporated into the implementation.

Conclusion

This chapter has described the UniFrame process for constructing distributed computing systems and has shown how it facilitates achieving the current goals of government and industry in rapidly creating high quality computing systems. UniFrame provides a framework within which a diverse array of technologies can be brought to achieve these ends. These include software engineering practices, such as rapid, iterative, and incremental development. Its business line, service-oriented, model-driven architecture based on components is a realization of the movement to provide mutability, quick development, and conservation of resources. A knowledgebase of component-based, predefined and tested designs for distributed computing systems, event traces for empirical testing, and quality of service prediction and calculation are tools it utilizes for increasing quality assurance. UniFrame decouples the requirements analysis and system assembly activities from the problem of collecting appropriate components published on the network. Its novel resource discovery service facilitates the efficient acquisition of components meeting stated specifications. It provides a mechanism for seamlessly bridging components of different models, such as RMI and CORBA, to support the construction of heterogeneous, distributed computing systems having platform-independent definitions. The UniFrame project is also investigating techniques and patterns related to using quality of service parameters during the design of components and integrated systems to create high assurance distributed computing systems.

Acknowledgments

This work was supported in part by the U.S. Office of Naval Research, grant N00014-01-1-0746.

References

Auguston, M. (1995). *Program behavior model based on event grammar and its application for debugging automaton.* In M. Ducassé (Ed.), Proceedings of the 2nd International Workshop on Automated and Algorithmic Debugging (AADEBUG'95) (pp. 277-291), Rennes: Université de Rennes.

Batory, D., & Geraci, B. (1997). Component validation and subjectivity in GenVoca generators. *IEEE Transactions on Software Engineering, 23*(2), 67-82.

Beugnard, A., Jezequel, J., Plouzeau, N., & Watkins, D. (1999). Making components contract aware. *IEEE Computer, 32*(7), 38-45.

Brahnmath, G. (2002). *The UniFrame Quality of Service Framework*. Unpublished master's thesis, Indiana University Purdue University, Indianapolis, IN, United States. Retrieved August 8, 2004: *http://www.cs.iupui.edu/uniFrame/*

Brown, A. (1999). Building systems from pieces with component-based software engineering. In P. Clements (Ed.), *Constructing superior software* (Chapter 6). Indianapolis, IN: MacMillan Technical.

Bryant, B. R., Auguston, M., Raje, R. R., Burt, C. C., & Olson, A. M. (2002). *Formal specification of generative component assembly using two-level grammar.* Proceedings of SEKE 2002, 14th International Conference on Software Engineering and Knowledge Engineering (pp. 209-212). Los Alamitos: IEEE Press.

Bryant, B. R., & Lee, B.-S. (2002). *Two-Level grammar as an object-oriented requirements specification language.* Proceedings of HICSS-35, the 35th Hawaii International Conference on System Sciences (p. 280). Los Alamitos, CA: IEEE Press. Retrieved August 8, 2004: *http://www.hicss.hawaii.edu/HICSS_35/HICSSpapers/PDFdocuments/STDSL01.pdf*

Crnkovic, I., Schmidt, H., Stafford, J., & Wallnau, K. (Eds.). (2003). Proceedings of the 6th Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction. The 25th International Conference on Software Engineering (ICSE). Retrieved August 8, 2004: *http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6*

Enterprise Architecture SIG, Industrial Advisor Council (IAC). (2003a, March). Business line architecture and integration. Retrieved August 8, 2004: *http://216.219.201.97/documents_presentations/index.htm*

Enterprise Architecture SIG, Industrial Advisor Council. (2003b, March). (IAC). Succeeding with component-based architecture in e-government. Retrieved August 8, 2004: *http://216.219.201.97/documents_presentations/index.htm*

Fritzson, P., Auguston, M., & Shahmehri, N. (1994). Using assertions in declarative and operational models for automated debugging. *The Journal of Systems and Software, 25*, 223-239.

Griss, M. L. (2001). Product line architectures. In G. T. Heineman, & W. T. Councill (Eds.), *Component-based software engineering: Putting the pieces together* (pp. 405-420). Boston: Addison-Wesley.

Heineman, G. T., & Councill, W. T. (Eds.). (2001). *Component-based software engineering: Putting the pieces together*. Boston: Addison-Wesley.

Huang, Z. (2003). *The UniFrame system-level generative programming framework.* Unpublished master's thesis, Indiana University Purdue University, Indianapolis, IN, United States. Retrieved August 8, 2004: *http://www.cs.iupui.edu/uniFrame*

Java Remote Method Invocation – Distributed computing for Java. (2003, October 2). Retrieved August 8, 2004: *http://java.sun.com/marketing/collateral/javarmi.html*

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21*(7), 558-565.

Lumpe, M., Schneider, J., Nierstrasz, O., & Achermann, F. (1997). *Towards a formal composition language.* In G. T. Leavens & M. Sitamaran (Eds.), Proceedings of the 1st ESEC Workshop on Foundations of Component-Based Systems (pp. 178-187). Heidelberg: Springer-Verlag.

Microsoft .Net Framework: Technology overview. (2003, October 2). Retrieved August 8, 2004: *http://msdn.microsoft.com/netframework/technologyinfo/overview/*

Object Management Group. Model-Driven Architecture™, the architecture of choice for a changing world. (2002, March 12). Retrieved August 8, 2004: *http://www.omg.org/mda*

Raje, R. (2000). *UMM: Unified Meta-object Model for open distributed systems.* Proceedings of the Fourth IEEE International Conference on Algorithms and Architecture for Parallel Processing (ICA3PP 2000) (pp. 454-465). Los Alamitos, CA: IEEE Press.

Raje, R., Auguston, M., Bryant, B., Olson, A., & Burt, C. (2001). *A unified approach for integration of distributed heterogeneous software components.* Proceedings of the Monterey Workshop on Engineering Automation for Software Intensive System Integration, SEAC technical report (pp. 109-119). Monterey, CA: U.S. Naval Postgraduate School. Retrieved August 8, 2004: *http://www.cs.iupui.edu/uniFrame/*

Raje, R., Bryant, B., Olson, A., Auguston, M., & Burt, C. (2002). A quality-of-service-based framework for creating distributed heterogeneous software components. *Concurrency and Computation: Practice and Experience, 14,* 1009-1034.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language reference manual*. Reading, MA: Addison Wesley.

Schmidt, D. (2003, October 2). Overview of CORBA. Retrieved August 8, 2004: *http://www.cs.wustl.edu/~schmidt/corba-overview.html*

Siram, N., Raje, R., Olson, A., Bryant, B., Burt, C., & Auguston, M. (2002). *An architecture for the UniFrame Resource Discovery Service.* Proceedings of the 3rd International Workshop of Software Engineering and Middleware: Vol. 2596. Lecture Notes in Computer Science (pp. 20-35). Heidelberg: Springer-Verlag.

Sun, C. (2003). *QoS composition and decomposition models in UniFrame.* Unpublished master's thesis, Indiana University Purdue University, Indianapolis, IN, United States. Retrieved August 8, 2004: *www.cs.iupui.edu/uniFrame*

Szyperski, C., Gruntz, D., & Murer, S. (2002). *Component software - Beyond object-oriented programming*. (2nd ed.). Boston: Addison-Wesley/ACM Press.

Warmer, J., & Kleppe, A. (2003). *The Object Constraint Language*. (2nd ed.). Boston: Addison-Wesley.

Weck, W. (1997, June). Independently extensible component frameworks. In M. Mühlhäuser (Ed.), *Proceedings of the 1st International Workshop on Component-*

Oriented Programming (European Conference on Object-Oriented Programming, Jyväskylä, Finland), *Special Issues in Object-Oriented Programming* (pp. 177-188). Heidelberg: Springer-Verlag.

Chapter V

Service-Oriented Design Process Using UML

Steve Latchem
Select Business Solutions Inc., Gloucester, UK

David Piper
Select Business Solutions Inc., Gloucester, UK

Abstract

This chapter presents a worked example of a design process for Service-Oriented Architecture. It utilizes the industry standard modeling notation, the Unified Modeling Language (UML) from the Object Management Group, to present a practical design for services. The authors have used their real world experience on many service-oriented projects to develop a design method using visual modeling to implement high quality services and service implementation. The chapter introduces a terminology for services and their implementing components and then works through the example to show how the implementation is designed in UML. We hope that this will show the reader how services are implemented by organizations on real projects.

Introduction

We have been assisting organizations to use Component-Based Design and Development to implement Service-Based Architecture for over seven years, utilizing the Select Perspective development process and the principles of the Supply, Manage, and

Consume Model for Web services and dependent components. Web services are a natural development from components, which in turn were a natural development from OO. We have leveraged the power of the Object Management Group's Unified Modeling Language (UML) to analyze and specify the services within a Service-Oriented Architecture.

In this chapter, we intend to initially present the difference in terminology from the Component View of the world and the Service view of the world. We will then present a worked example of the process and modeling *hotspots* for implementing Web services and Component Based Development:

- **Business Process Modeling:** the identification and definition of the business processes, including their inputs, outputs, and dependencies
- **Business Web Page Design:** the identification and modeling of the Web page designs and their interaction within the Business Web
- **Web Service Identification and Reuse, Supply, Manage, and Consume (SMaC):** defining the requirements and prospecting for reusable Web services to support the Business Web Pages and the Business Processes defined
- **Web Service Internal Design:** designing and constructing the components and agents that will deliver the component functionality to execute the required Web services across the distributed domain
- **EAI:** Enterprise Application Integration, effectively "hooking" in the requests to both legacy/package data and functionality to deliver the required Business Processes across the software architecture that is currently in place
- **Testing:** providing the capability to dynamically build the test cases and scripts from the design environment and track the test results

Component View

Component Based Development (CBD) has gained great popularity in recent years as the technology required to support the development and use of components has matured. Highly capable modeling environments, such as Select Component Factory, support the CBD process from business alignment through to solution deployment by fully implementing the SMaC paradigm.

CBD promises the key benefits of high levels of reuse and complete interoperability between different forms of component implementation as long as they share a common communications standard. Implementations for components include the use of:

- Object technology to implement new business functionality;
- Heritage code wrapping to reuse existing functionality;

- Data wrapping to separate data manipulation from data persistence;
- Package wrapping to support the integration of package and bespoke code; and
- Purchase or hire of services from open service markets.

Despite these varying component implementations, solutions which make use of the business services provided by components see only components and the business services they offer. All functionality is provided to the solution via the components' business services. The homogeneity of this architecture is a key benefit to the solution. In particular, it supports the plug-and-play concept, allowing current component implementations to be replaced in a manner that is transparent to the solution and, in consequence, to the user. From the solution developers' point of view, the simple architecture simplifies and speeds development.

Figure 1 shows the external, published specification of a component, *AccountCustomer*. The component offers two *service interfaces: ICustomer* and *ICustomerAccount*. Each of the service interfaces offers *business services* to one or more business solutions. Also shown in the figure are two business solutions that make use of the interfaces offered by the *AccountCustomer* component. The assembly of the business services to provide the behavior required by the solution can be modeled using UML interaction diagrams.

Figure 2 shows a partial implementation for the component, including the service interfaces in noniconic form. Each operation listed on the interfaces represents a

Figure 1. Component specification

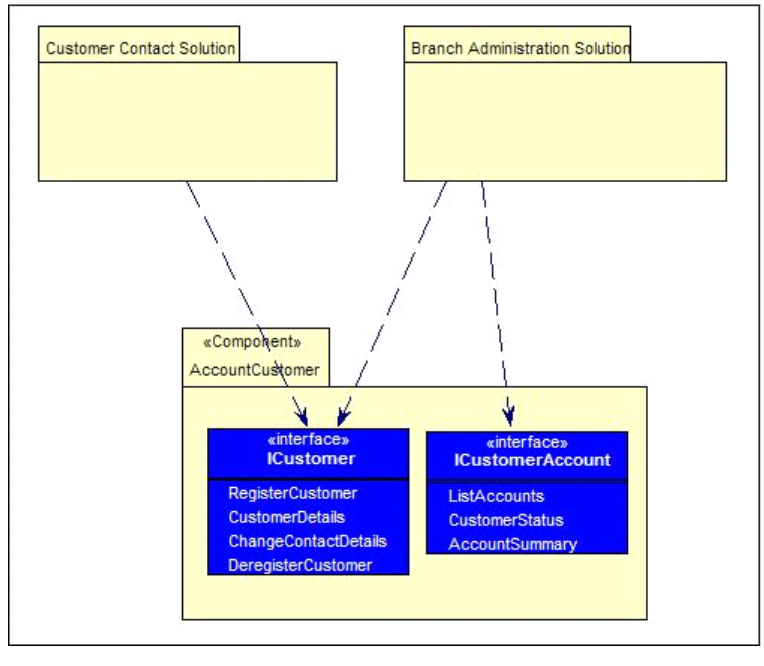
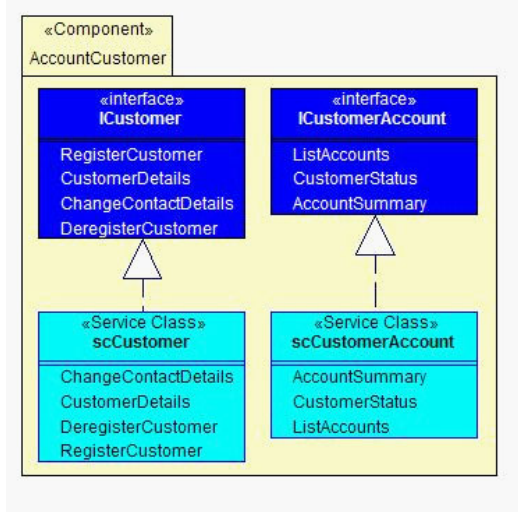


Figure 2. Partial component implementation



business service that the service interface offers. The internal architecture of the component shows that a *service class* implements each of the service interfaces. The service class will be responsible for delegating the service request received by the component onto the implementation constructs (classes, legacy code, and so forth) that exist within it. These additional implementation artifacts have been omitted.

Component Terminology

Together these show examples of the key terms commonly used in the component-based world. They do not represent a complete set of terminology; for example, stereotypical architectural roles can be drawn out for different types of components (see Table 1).

Web Service View

Web services are seen to be effective mechanisms for providing functionality in widely distributed architectures, including typical Business-to-Business (B2B) applications. Their key feature is the very weak coupling required between the solutions using the Web services and the Web services being used. The use of a directory to resolve the location of the implementation of a Web service is a powerful feature, supporting the runtime coupling of solution to Web service. Directories also potentially offer the capability for

Table 1.

Term	Definition
Component	A mechanism for offering multiple business services to one or more business solutions; a unit of deployment for the implementation.
Service Interface	An interface onto a component – a component will offer one or more service interfaces.
Business Service	An operation defined on a service interface that supports some need expressed by one or more business solutions.
Service Class	A class that provides the implementation of a service interface on a component. The service class receives the service request and delegates control to the artefacts internal to the component.

automatic fail-over so that in the case of a service provider being temporarily unavailable, alternative implementations from other providers can be invoked.

There is now a strong argument about the identity of the Web service content. Are Web services fundamentally predicated on the Web? Must Web services use the current standards of SOAP, UDDI, WSDL, WSCL, and so on? More importance is being given to the view that Web services are not about the standards used for any specific implementation but more about the concepts of a defined interface and protocol for the service itself. Indeed, even the term *Web* is being increasingly dropped from their name. Weak coupling through a directory is seen as a key differentiator between components and services in a service-oriented architecture.

Figure 3. External view of Web services use

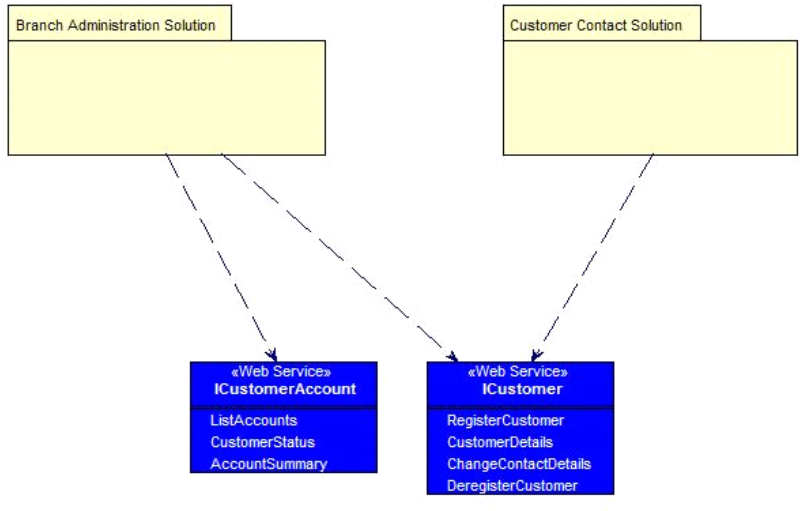


Table 2.

Term	Definition
(Web) Service	An interface defining the protocol of the service – it infers nothing about the implementation of the service – indeed each service may be backed by several, distinct implementations, perhaps using different technology.
(Web) Method	An operation defined on an interface that supports an aspect of the protocol of the service.

Figure 3 shows the external view of the use of Web services by solutions. Each of the Web services is available independently and there is no information given about their implementation — whether as distinct executables or grouped together as a component. Indeed, the loose coupling of the directory implies that the services may be implemented by different suppliers, so no information concerning their physical configuration (other than that provided in the directory) can be inferred.

Each of the *Web services* is implemented via a defined interface offering a number of *Web methods*. The current standards explicitly support the concept of multiple implementations of a single Web method being supported.

Service Terminology

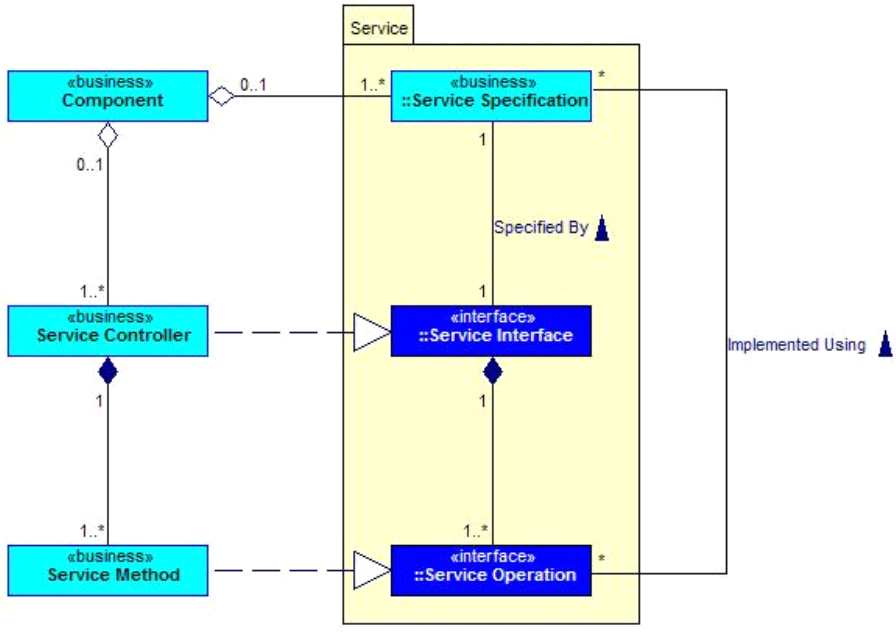
This shows examples of the key terms commonly used in the service-based world. They do not represent a complete set of terminology; for example, stereotypical architectural roles can be drawn out for different types of components (see Table 2).

A Service-Oriented Approach

Is it possible to define an approach and terminology that effectively unifies the component and Web services viewpoints? Such an approach is defined by a *service-oriented architecture,* a solution architecture that offers functionality to the user only through services. The services may be implemented by components — newly built, bought-in, legacy wrapping — or as Web-style services.

From each solution's point of view, the environment looks completely uniform; all functionality is exposed as services provided by interfaces. Technical complexities around the implementation of those services are hidden from the user (or encapsulated) in exactly the same way as they are in both the component and Web service worlds. Indeed, we can imagine complex architectures where Web services are wrapped inside components and are, in turn, implemented by components in the target environment.

Figure 4. Service-oriented terminology



The Role of Components

In the service-oriented architecture, the role of components as a deployment mechanism is reduced since at least some of the functionality is going to be delivered in other ways. What, then, is the role of the component?

Certainly components are still a key deployment route. Performance critical functionality and functionality that encompasses key competitive advantage will still be retained in-house implemented as relatively tightly coupled behavior. In development process terms, components continue to play other important roles. They remain the key vehicle for the early identification of groupings of information and behavior and, in consequence, serve as the vehicle for the provisioning of the functionality. It will often be at the point of making provisioning decisions that the choice between component and Web-service implementations will be made. The choice will be influenced by commercial and technical constraints but does not need to be predetermined as part of the analysis process.

At the specification level, when analysis techniques are primary, the service-oriented architecture is still dominated by components and their interfaces. Driven by the selected architectural constraints, design and implementation takes advantage of the extra

Table 3.

Service-Oriented Term	Definition	Component Term	Web Service Term
Component	A container used for the analysis of the business architecture, encapsulation of functionality and information and as a deployment vehicle for component-based services	Component	None
Service	The combination of the specification, interface and set of service operations that together define a service	Component Specification	None
Service Specification	The specification of functionality to be offered to one or more consumers	None	Web Service
Service Interface	An interface offering a protocol capable of providing the specified functionality to one or more consumers	Service Interface	Web Service
Service Controller	A mechanism for implementing the service interface	Service Class	None
Service Operation	The specification of a unit of functionality required to execute the service; forms part of the protocol of the service	Business Service	Web Method
Service Method	The implementation of a service operation by a physical method on the service controller	Service Class Method	None

flexibility offered by the service-oriented architecture to choose the most appropriate implementation and deployment mechanism.

Service Oriented Terminology

The concepts and related terminology used in the definition of the service-oriented architecture are illustrated in the meta-model shown in Figure 4.

The concepts shown are defined in the Table 3. Related component and Web service terms are offered for comparison.

Stereotypes

Within the service-oriented architecture clear stereotypes of the specific concepts can be identified. The stereotypes are derived from both the component-based and Web service origins of the service-oriented architecture. These stereotypes are used to enhance the UML specification, and are typically added to Packages, Classes, and Operations to specify Service-Oriented Architecture concepts to these UML artifacts. Example stereotypes are defined in Table 4.

Table 4.

Stereotype	Definition
«Business Component»	A component focused on offering one or more services that manipulate business information and implements business policies, rules and algorithms.
«Business Service»	A service (potentially implemented by a «business component» or as a «web service») that specialises in manipulating business information and implements business policies, rules and algorithms.
«Data Service»	A service responsible for accessing and updating data in persistent storage within the context of a transaction.
«Data Service Operation»	A service operation, typically offered by a «data service», responsible for performing a specific persistent storage operation.
«Process Component»	A component focused on the implementation of a process by controlling the protocol of service operation invocation on services.
«Web Method»	The implementation of an operation defined by the service interface of a «web service».
«Web Service»	A service implemented using internet protocols and which adheres to specific standards for service location through a service directory.

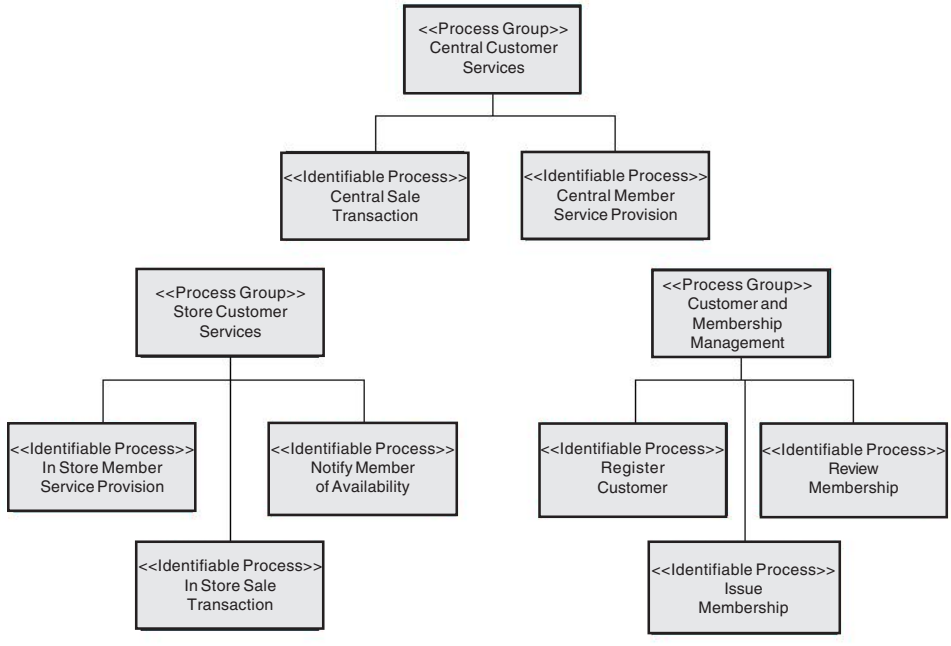
Worked Example

The following sections provide an example of the Select Perspective Web Services and Components Process and Modeling Artifacts. To usefully demonstrate this, we have used a candidate Business Web, consisting of a number of organizations looking to provide a Consolidated Product Catalog via a Web application, utilizing the latest Web technologies and allowing customers to purchase products from that catalog. We have called our "virtual" Business Web Organization *Web Entertainment Products*. The Web Entertainment Products organizations' business is the sale of products; we have assumed these are leisure media products, for example, CDs, computer games, and books. A group of organizations in this space have come together to provide a consolidated offering of rental and sales of these items to a broad customer base. The sales portal will be their consolidated Web site with the execution of new and existing application functionality making the catalog available to browse and complete a rental or sale transaction for the selected product, irrespective of the organization within the Business Web that supplies it.

Business Process Modeling

Business Process Modeling (BPM) is a precursor technique used to establish the content and scope of the business domain before Solutions Design begins in earnest. It establishes a shared understanding of the business domain expressed in a formal manner. By focusing on business processes, the model can be used as a starting point to drive synchronized programs of business and systems change. The new business processes

Figure 5. Process hierarchy diagram



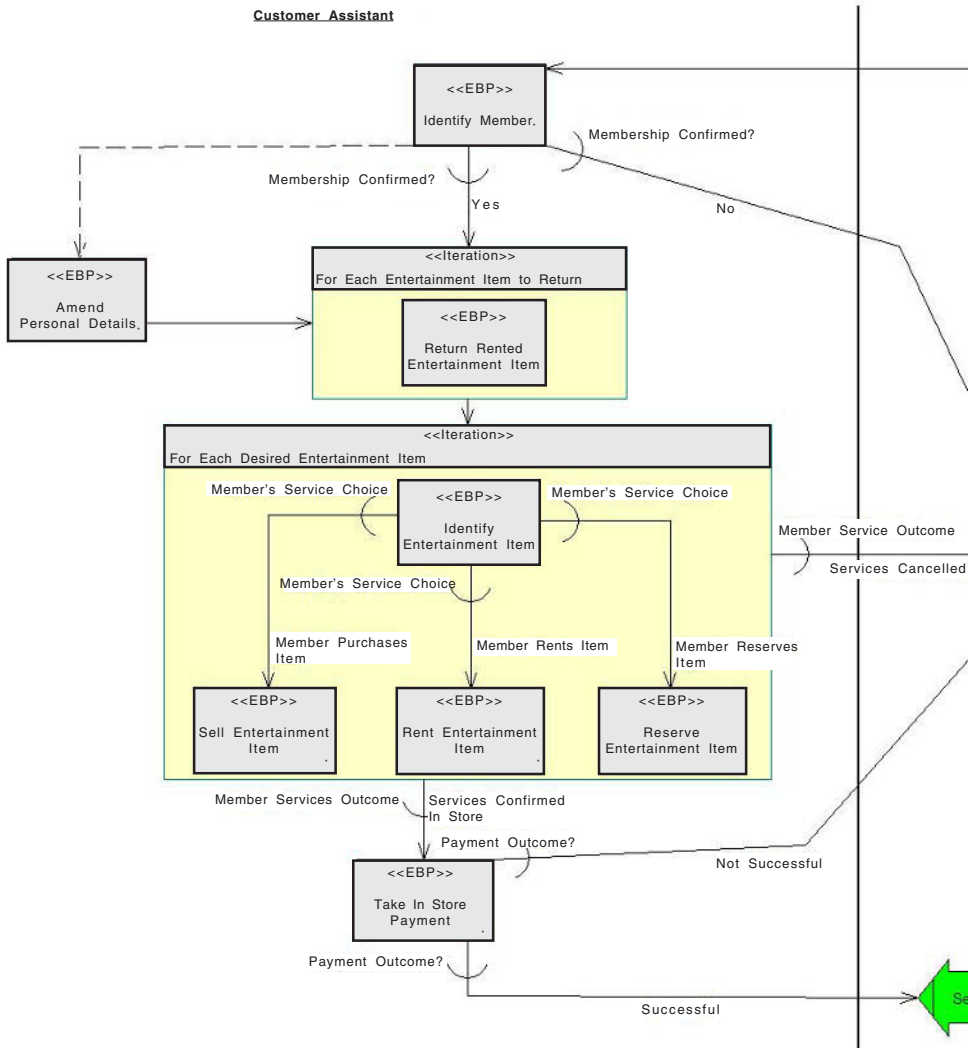
must be implemented both by the business and used as requirements for constructing new solutions to support those processes.

Web services standards are taking an increasingly abstract view of the way that organizations and solutions will interact. Currently, the most abstract view is at the business process level. Standards such as BPML (*http://www.bpmi.org*) allow organizations to publish the structure and protocols of their business processes. Within these published definitions, the requirements and implementations of Web services together can be published.

One critical aspect to any Service-Oriented Architecture is the need for Business Alignment. The exposure of services from existing applications is not enough; services need to be delivered to a business context for the organization today and its future processes and capabilities.

Here we use the industry-leading Business Process Modeling notation from Computer Sciences Corporation, Catalyst. The notation is supported by a large number of CASE modeling tools, for example, Select Component Architect, System Architect, Aris, and CaseWise.

Figure 6. Process thread diagrams

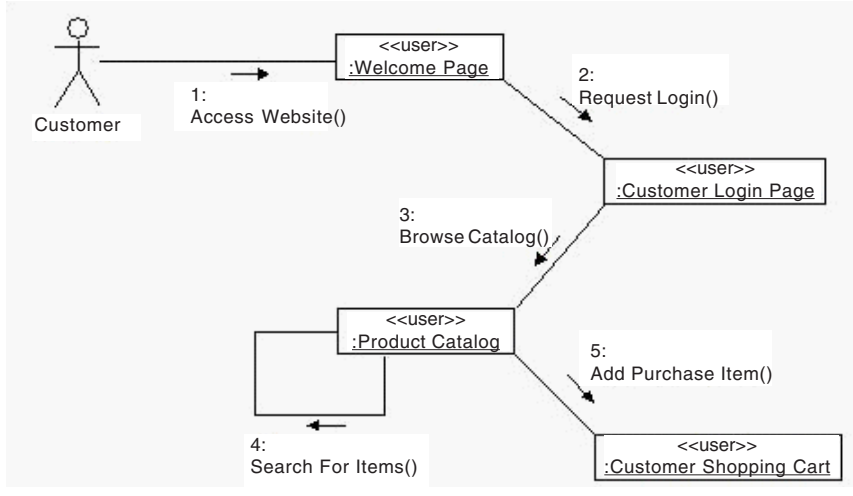


The notation has two key diagramming types:

- **Process Hierarchy:** to present the hierarchical structure of business process groups (nonelemental processes);
- **Process Threads:** to show the dynamic transitions, constraints and input/outputs of elementary business processes.

The hierarchy diagram in Figure 5 shows the hierarchical relationships between process groups for Web Entertainment Products. These are the Stock Management, Online

Figure 7. Business Web page design



Transactions/Customer Services, and the Registration and Acceptance of Customers/Members.

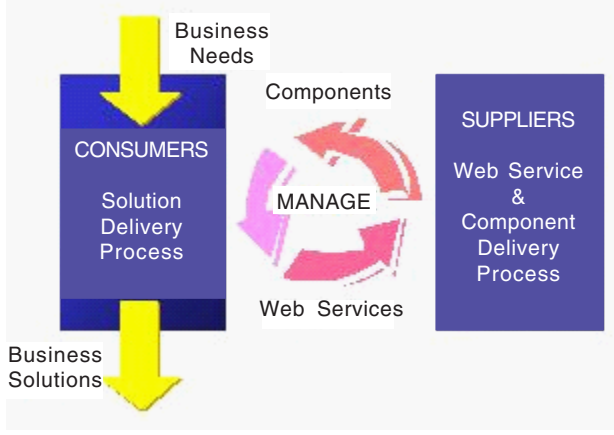
For the thread diagrams in our example, we have concentrated on the customer transaction, encompassed within the Process Group – Online Customer Service Provision.

The thread diagram in Figure 6 shows the Events and Results of the Process Group that is its parent, their transitions, exclusivity (represented by the arcs on the transitions), and the elementary business processes (EBPs) to be implemented. In addition, *swimlanes* are added to represent the ownership of each EBP by Business Actors, in this case, the customer using the Web sales portal. Documenting each EBP are detailed *descriptions*, *objectives* and *process volumetrics*, enabling the designer to not only refine and/or refactor the business processes before automation but also to look at *performance and scalability* of nonfunctional requirements. In addition, traceable links to use cases allow the dependencies between the business processes and systems requirements to be understood easily.

In parallel with the Business Process Modeling, the "front of store" work is typically carried out. The general graphic user interface design, impact/atmosphere, core page prototyping, and market analysis are performed. Here we use Collaboration Diagrams defined by the Unified Modeling Language (UML). This has two purposes:

- To design the *page interactions and dependencies,* in the context of the actor using the Web solution;

Figure 8. Supply, manage, and consume



- To scope the pages to be built, associate the growing *graphic designs* of these pages, and support the development of the *Use Cases* to define the *Web Service* requirements.

The Object Collaboration Diagram in Figure 7 shows the simple scenario of Logging In to the Web site, browsing the entertainment item product catalog, and adding an item to an Internet shopping cart for rental/purchase later. While this is not a User Interface prototype, this clearly shows the involvement of the Server Pages, and their interaction for this scenario.

Web Service Identification

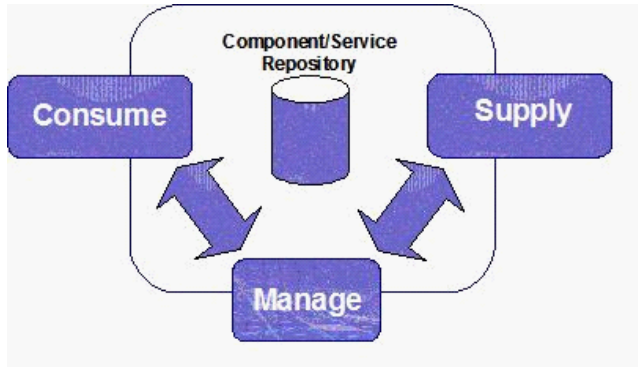
The next stage of the Business Web life cycle is to identify and define the Web services that will fulfill the requirements of the Business Web. This includes the reuse of Web services from external parties (that is, commercial UDDI repositories) or internal sources (that is, the internal repository of Web services and components) that may fulfil the Business Web process requirements.

Typically, an organization would use the Business Process definitions to browse and identify potential Web services/components then include these into this modeling phase after performing a gap analysis.

Supply, Manage, and Consume Model (SMaC)

The diagram shows the SMaC model for components and Web services, where suppliers (external or internal) and consumers form a reuse relationship for the delivery of the

Figure 9. Component and service repositories



Business Web solution. The central Manage function deals with publication of service/component versions, the service/component consumer lists, and the dependencies between services and components. This enables the organization to employ a solution, service and component "factory" model in their analysis, design, reuse, and construction of software solutions.

Figure 8 shows a high-level graphical representation of SMaC, where Business Needs feed into a Consume process, services are sought through the central Manage function, and delivered through the Supply function; then, the deliverable from Consume is an integrated Business Solution to fulfil the Business Needs.

Identification and reuse of services is a practice already well established in the CBD arena, especially in the concept of Supply-Manage-Consume. The maturing Component Management tools enable publication, search, and reuse of components/services.

Consumers can publish the specification of new components and services as requests for delivery. They consume existing components and services as they assemble or reassemble solutions. Consumers are typically not interested in the implementation technology of a particular service as long as it meets the functional and nonfunctional specification and can integrate with their solution architecture.

Suppliers receive component/service specifications and supply implementations in return. Different sources of components are maturing rapidly; in-house teams are increasingly skillful at component delivery; System Integrators have adopted component supply as a valuable channel of business; Package solutions are becoming componentized; Component Market Places are maturing rapidly. Web services provide another channel for the supply of services.

The identification and definition of Web services forms a critical part of Business Web Design, particularly when the power of Web Service Discovery technology (UDDI) is to be used (*http://www.uddi.org*). This uses a Web Service Business Registry to lodge and discover Web services.

Web Services Provisioning is a variation on a theme. The need is the same as for other forms of provisioning: to identify services that meet a set of functional, informational, and protocol requirements. In common with other channels of provisioning, a common medium for stating the need has been specified — UDDI. This is different in form but not in fundamental nature from specification mechanisms used by other channels. There are additional elements to consider, particularly if Web services are from third parties, the aggregation of Web services to support the transactional context required by the organization's business processes, and the security/trust facilities provided by the service. Nonfunctional requirements are considered here, too. For example, does the service provider guarantee sufficient robustness, scalability, availability, and performance?

Having identified candidate services, details of these can be published by the Component Management function into an organizations component/service repository for future reuse. Activities performed by the Component Manager in relation to Web services are similar to those carried out for any other form of service, including Certification, Quality Assurance, Reuse Advice, and Authorization. The Web services model is essentially commercial at base, but in SMaC focused organizations, other commercial service channels are already in use, including Component Market Places. While the economic model may differ in form, the need to work within a commercial licensing framework is not new.

Figure 10. Use case model

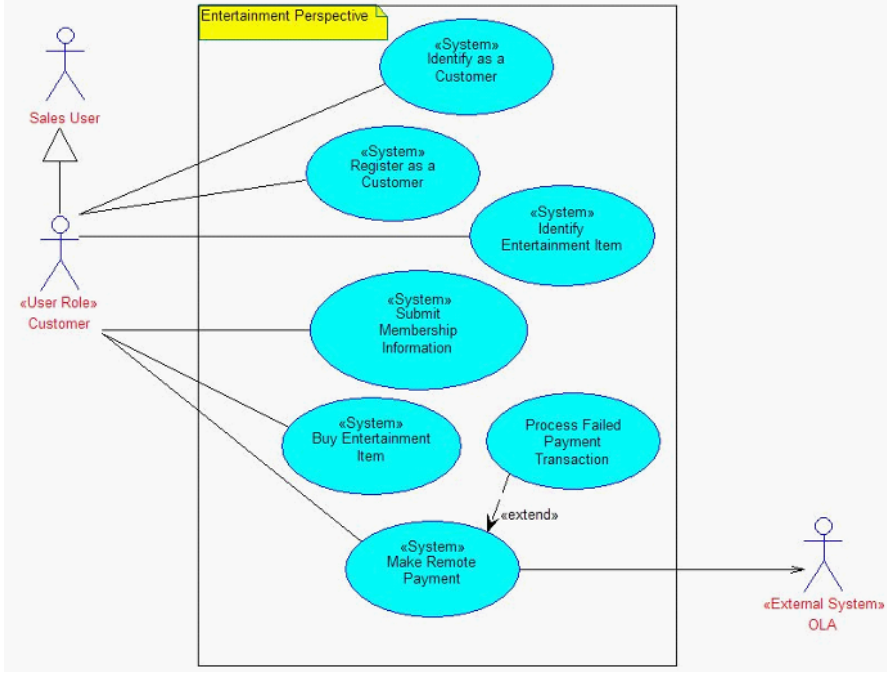
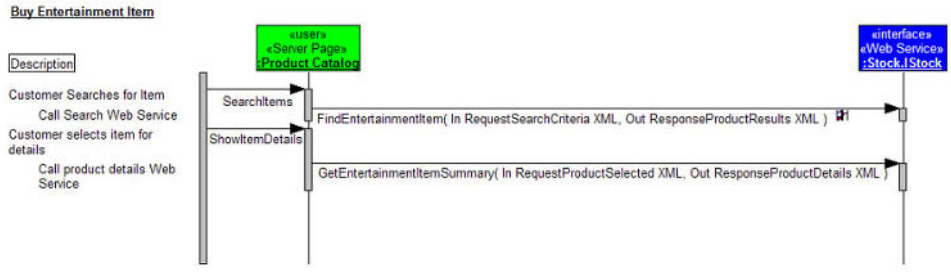


Figure 11. Service level Object Sequence Diagram



Gap Analysis and Business Web Design

For this analysis and design, SBS uses the power of UML. Use Cases are used to define the functional requirements of the system in terms of the Actor's interactions with the system. These Use Cases are directly mapped to the Elementary Business processes they have been derived from, providing *complete traceability* to the Business Model.

The Use Case diagram in Figure 10, defines the Use Cases for the Customer system actor; it also shows the use of an external System Actor for Online Authorization.

From the Use Cases, the Web services can be identified and modeled using Object Sequence Diagrams to define the contextual use, and definition of the Web Services. Each Use Case has a detailed Intent, Description, Pre/Post Conditions and Alternate Course descriptions, enabling the Designer to identify and define the Web services in detail.

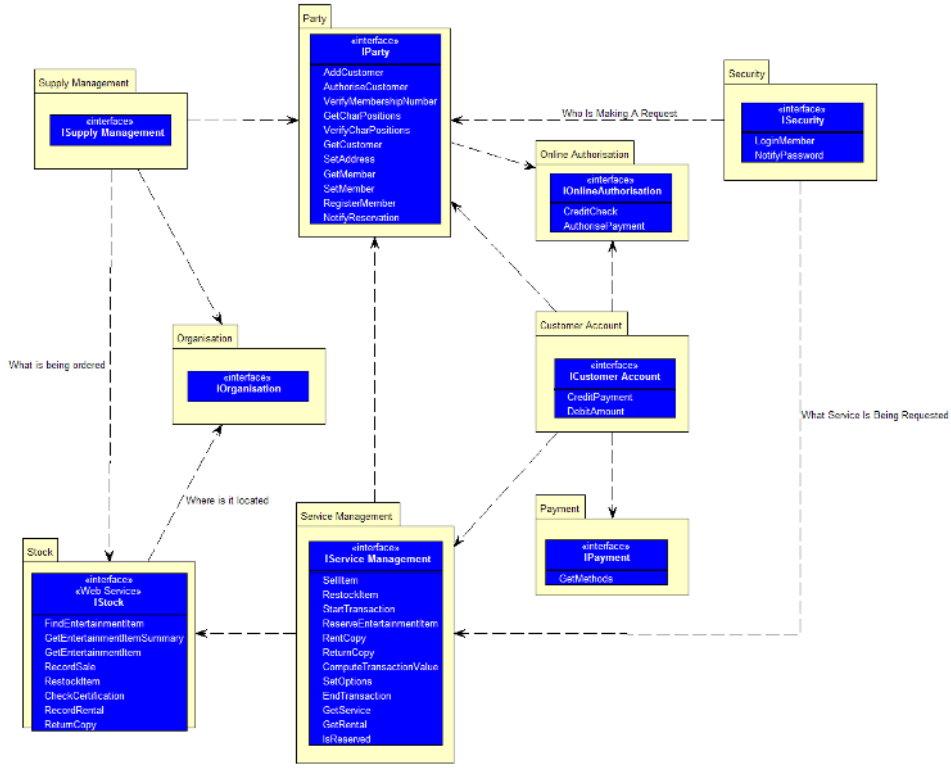
The Object Sequence Diagram in Figure 11 shows the Web service identification from the Identify Entertainment Item Use Case. Note that the Services defined follow the slightly different paradigm-to-component services, that is, they are coarse grain, *single shot* service requests that perform a single focused business service for the calling Web solution.

The Web services are *bundled* into cohesive groups, defined as UML Packages to show their notional inclusion in virtual components. Additionally, the internal data/XML structure of the collection of services is modeled as class diagrams with relationships to show hierarchy and multiplicity.

The data-centric nature of a Web service contract, its input/output, drives the strong need for detailed data modeling of the XML structures. This effectively becomes the basis for information exchange internal to the organization and across organizations, similar to the data exchange and information modeling requirements for EAI and B2Bi initiatives, for example. This XML modeling (as Class/Package structures) extends the behavioral nature of UML modeling to include the data requirements for the services.

Figure 12 shows a Package Diagram of the *bundling* or *chunking* of the services to cohesive business groups, for example, Party for customer and member processing,

Figure 12. Web services Package Diagram



Organization for the company's services, and Online Authorization for the external use of a card processing service.

In Figure 13, we show an internal data model developed for each of our logical services groups. This is done to model the data structures and enables us to synchronize our XML design/implementation with the internally owned data for the service groups.

Web Service Internal Design

The supplier of the Web service, whether internal or external to the organization, now needs to design its concrete implementation. There are distinct layers to the implementation of a published Web service:

1. **Components and Agents:** the use of components and peer-to-peer (P2P) agents design to provide the dynamic deployment and discovery of components that will implement the Web service.

Figure 13. Internal Data/XML Structure Diagram

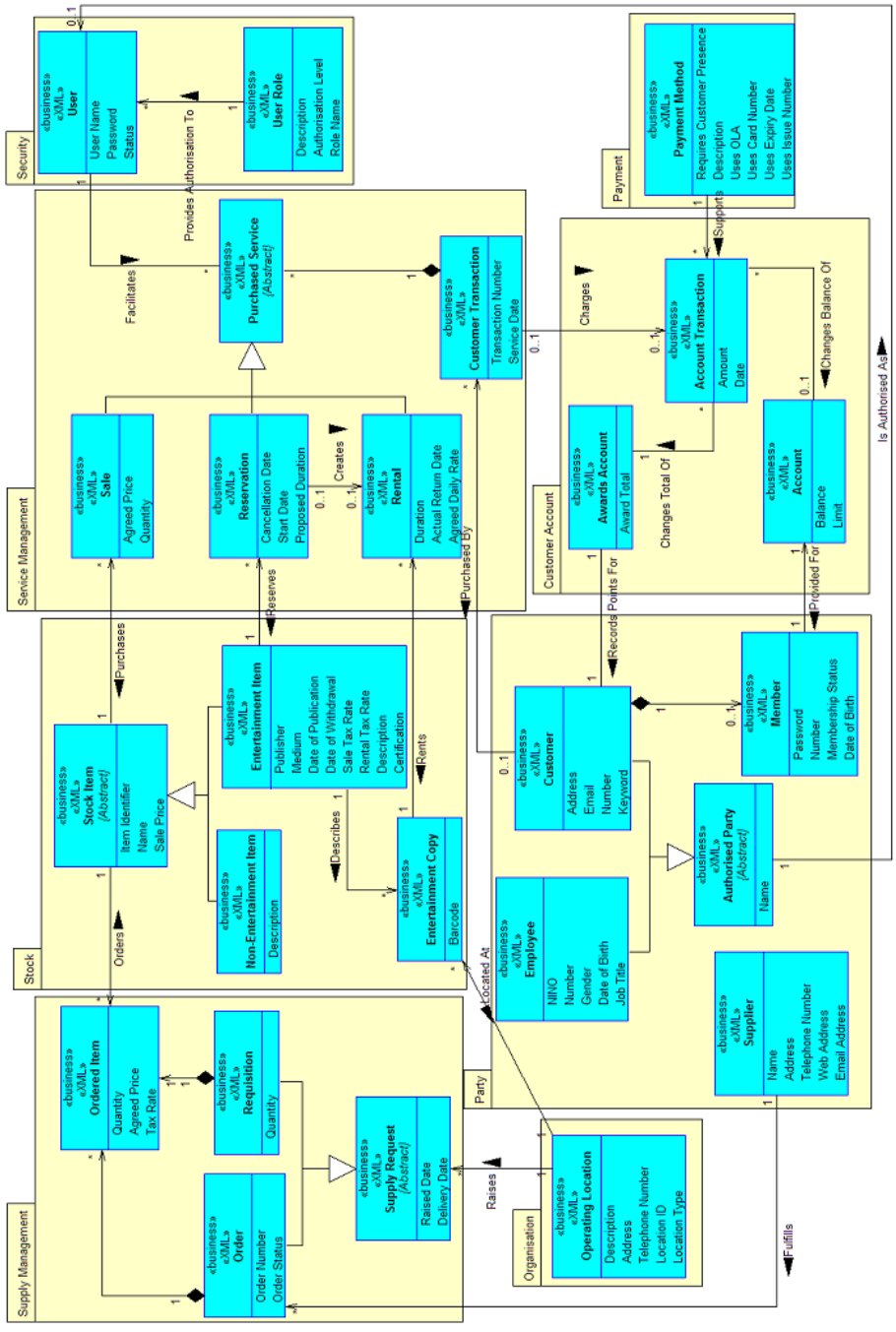


Figure 14. Internal service Object Sequence Diagram

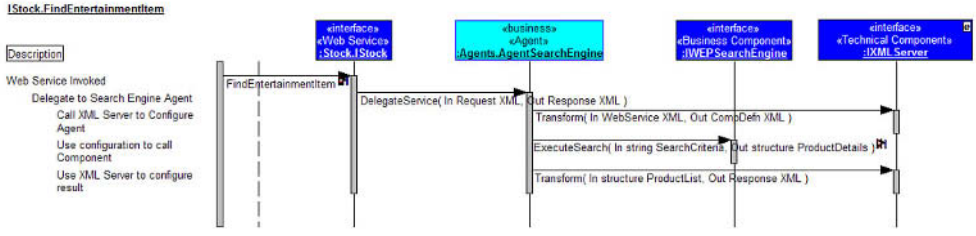
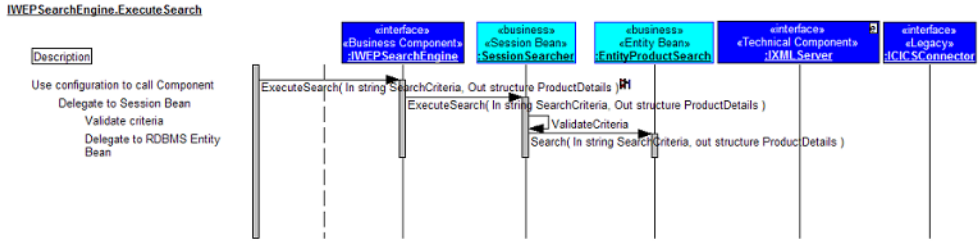


Figure 15. Internal Component Sequence Diagram



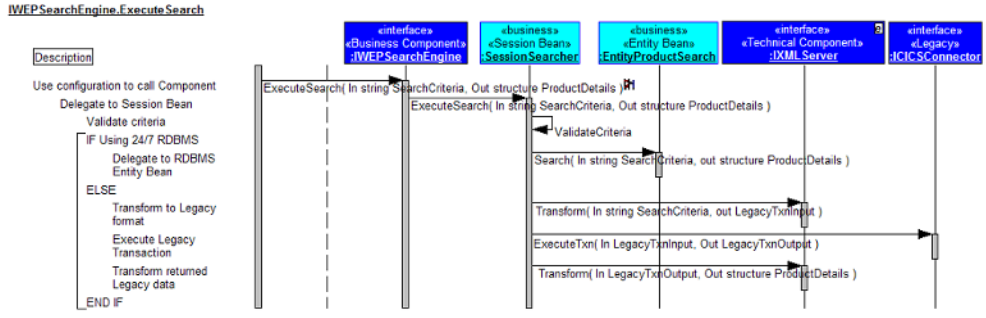
2. **Component Internals Design:** the internal design of business components that provide services to the Web service and the controlling agents.
3. **Legacy Integration (EAI):** the integration of legacy functionality and data into the Business Web internal components.

Component and Agent Design

For each Web service designed, its internal implementation is now required. The Object Sequence Diagram in Figure 14 shows the design of the inclusion of this agent and its use of the interface to an XML Server to configure its actions when the Web service is requested.

The business components themselves, like our Search Engine from the Figure 14 Object Sequence Diagram, require design, too. The Object Sequence Diagram in Figure 15 shows the decomposition of the component's internal object-oriented design, using a façade Session EJB, which delegates to an Entity EJB to perform the product search.

Figure 16. Internal Component Sequence Diagram with Legacy Wrapping



Integration Legacy Applications and Data – EAI Design

The final piece of Business Web design/development activity is the integration of legacy applications and/or data into the design of the Web service. For this integration work, Enterprise Application Integration (EAI) vendors typically provide solutions to seamlessly integrate the legacy systems. Typically, organizations make use of legacy application and system *Connectors*. The inclusion of these Connectors is modeled in the Object Sequence Diagrams to show the interaction and transformation of the component internals to the legacy message through the Connector.

In Figure 16, we show the internal design of a communication to a legacy procedure using a Connector and the required data transformation to turn our XML into a format understood by the legacy transaction.

Testing

The final stage of the Business Web development life cycle is testing. The designs that have been discussed so far provide an excellent documentation base for test cases and scenarios. The UML models are converted to test cases using a bridge. In the testing tools, these definitions of Business processes, Web services, components, agents, classes, methods, and data marshalling are refined to include the Business Inputs and Expected Results. As the Testing Tool records the test case interactions for repetition, the results are captured in its repository, but the basis of the test case has come automatically from the design. Additionally, the volumetrics captured at the Business process level can now form the basis of the acceptance criteria for the load/volume tests and the deployment and load balancing tests that are performed.

Summary

For the CBD-experienced organization used to assembling solutions in a service-based architecture, Web services are an interesting new channel of service provision. They present no fundamental challenge in terms of analysis or design. Nor do they present a challenge in terms of processes and structures for managing the virtual software assets. CBD-mature organizations are already assembling solutions from a mixture of software assets, both created internally and provisioned from external sources.

The challenge is in adapting to the details of the commercial models underlying Web service usage and in extending the existing provisioning mechanisms, so they support the active search for and publication and reuse of Web services. With Select Perspective, a mature Supply, Manage, and Consume (SMaC) model aligns perfectly with the principles of Web services and components, enabling the organization to migrate to Web services processes and architectural models more easily.

References

Apperly, Latchem, McGibbon, Piper, Maybank, Hofman, Simons, Service- and Component-based Development, Using the Select Perspective and UML, ISBN 0-321-15985-3

http://www.selectbs.com/resources/resources.htm - various White Papers and collateral relating to methodology and visual modelling for services.

Chapter VI

Service-Oriented Computing and the Model-Driven Architecture

Giacomo Piccinelli
University College London, UK

James Skene
University College London, UK

Abstract

Service-Oriented Computing (SOC) and the Model-Driven Architecture (MDA) are complementary systems development approaches with the mutual aim of reducing the cost of future systems integration. This chapter introduces the MDA concept and technologies to an SOC audience and employs these technologies to enhance support for SOC through the definition of a domain-specific modeling language for electronic services. The language is defined as an extension of the Unified Modeling Language (UML). Its semantics are defined using a domain model of electronic service systems based on concepts drawn from literature and experience with a range of commercial platforms for the deployment of electronic services.

Introduction

Service-Oriented Computing (SOC) and the Model-Driven Architecture (MDA) are both approaches to developing systems that anticipate the need for integration in heterogeneous computing environments.

SOC attempts to lessen the cost of future integration by making a recommendation about the *design* of systems: a service paradigm should be applied to software and business functions to support standardized communication and control technologies, such as Web services and workflow languages.

The MDA attempts to lessen the cost of future integration by making a recommendation about the *process* by which systems are developed: systems should first be developed as abstract models that do not contain technical details related to implementation, then transformed into platform-specific representations. Removing the need to disentangle business functionality from legacy platform decisions when redeploying all or part of a system on a new platform reduces the cost of integration when it is necessary to support a previously unexpected integration technology, such as a new middleware. As the MDA matures, the cost of redeployment may be reduced still further by the availability of reusable automated transformations to produce platform-specific models from platform-independent models.

The MDA is potentially a good complement to SOC. Service-oriented systems can be developed according to the MDA process in order to structure a system according to a services paradigm while maintaining a platform-independent specification. The first objective of this chapter is to introduce the MDA concept and its supporting technologies to an SOC audience.

The second objective of this chapter is to discuss how MDA standards such as the Enterprise Distributed Object Computing (EDOC) profile support SOC. The EDOC profile defines standard extensions to the Unified Modeling Language (UML) to allow the platform-independent modeling of enterprise computing systems, a class of system that subsumes electronic services.

The third objective of this chapter is to show how MDA support for SOC can be fruitfully expanded. The MDA approach gains productivity advantages when supported by domain-specific languages, such as the EDOC profile, for modeling systems in particular application areas. The advantage of such languages is that they allow the modeling of systems in a manner that is more concise and less error prone than if attempting to model the same systems using a more generic language. The EDOC profile does not allow the explicit modeling of several system facets unique to electronic services. We therefore believe it beneficial to provide more refined support for modeling electronic services by defining a UML extension (a profile) specifically for this purpose.

Our profile supports three modeling tasks particular to the development of electronic service systems:

- First, it allows the modeling of services at an abstract level, using the service vocabulary of capabilities, content, provisioning, offers, and information exchange. This allows the planning and documentation of the intended behavior for both single and coordinated services, essential when applying a service paradigm to systems development and a precursor to implementation efforts.
- Second, the profile allows the modeling of service deployments in terms of the concrete business assets that fulfil capability roles. This provides a concrete view

of services in the environment in which they will operate, supporting planning of service deployment and asset management.

- Third, the profile allows the modeling at a high level of abstraction of IT infrastructure components used to support the management and implementation of services. This includes Electronic Service Management Systems (ESMS), a class of application designed explicitly to support electronic services. The concepts underlying the profile are drawn in part from experience with a range of ESMSs. The identification of ESMSs and other infrastructure components, such as databases and workflow engines, in the profile reflects the reality of the electronic service world, in which services are seldom implemented from scratch but instead depend, to a large extent, on specialized platforms and predeveloped components.

Domain-specific modeling languages must be supported by strong semantic definitions so that the contribution of models to system specifications can be understood by developers and model transformations can be defined that produce platform-specific models according to the original intent of the developer as expressed in a platform-independent model. A popular approach to defining the semantics of an MDA language is by reference to a domain model, expressed in UML with accompanying natural language descriptions, which describes the concepts and relationships referred to by language constructs.

Our profile is defined with reference to a domain model of electronic service systems, based on an analysis of existing concepts from SOC literature and the author's experience of a range of ESMSs from major vendors.

The chapter is structured as follows: In the next section, we present an overview of the MDA and UML concepts with a particular emphasis on modeling using domain-specific language extensions. We then briefly discuss the EDOC profile and its application in implementing SOC systems. In the second section, we introduce our own domain model for high-level modeling of SOC systems, and in the third section, we define its associated UML extension. Subsequently, we present an example application of our UML extension to model services in the freight domain. In the final section, we discuss future trends and conclusions.

Background

The Model-Driven Architecture (MDA)

The MDA is an initiative of the Object Management Group (OMG), an industrial consortium chartered to standardize specifications for interoperable enterprise computing systems. In its early years, it standardized and championed CORBA, the Common Object Request Broker Architecture, a middleware platform that raised the level of abstraction for designing distributed systems by providing a number of critical *transparencies* for developers. Crucially, these included location and implementation trans-

parencies, meaning that components in a distributed system could be accessed using the same mechanisms regardless of their position in the network and the technologies used to implement their functionality.

Unfortunately, the success of CORBA was limited. To gain the advantages of CORBA, developers must choose to support it. Competition from other middleware standards such as Microsoft's COM and .NET and lately, the Web services initiative, have reintroduced heterogeneity at the middleware level.

In response, the OMG have made the decision to raise the level of abstraction still further. The Model-Driven Architecture (MDA) is an approach to systems development that aims to reduce the cost of deploying system functionality on multiple technical platforms, even after an initial implementation is complete. This cost reduction makes systems cheaper to maintain as technological standards change and easier to integrate either by migrating parts of the system to make use of a new integration technology or by reusing models of data and interactions in technology bridges. The following discussion of the MDA is drawn primarily from the MDA guide (OMG, 2003).

Fundamentally, the MDA approach consists of three recommendations concerning the process by which systems should be developed:

1. System designs should be expressed using models (typically UML models).
2. Models of two distinct kinds should be developed: Platform Independent Models (PIMs), which represent system functionality independently of features peculiar to any intended deployment platform and Platform Specific Models (PSMs), which provide a view of all or part of the system deployed using a particular platform technology and are detailed enough to be automatically converted into platform artifacts.
3. PIMs should be developed first and then automatically transformed as far as possible into PSMs.

These recommendations contribute to the goal of reducing the cost of redeploying system functionality in the following ways:

- System modeling ensures that designs and data models are preserved independently from implementations, allowing maintenance and integration to be performed without the effort of reverse engineering systems at the source-code level.
- Isolating models of business logic from the details of particular technical solutions makes this logic easier to reuse because it is easier to understand. Designs cannot depend on features or behaviors that are implicit to a platform, such as the operation of standard libraries or transactional behavior. Designs can be expressed at a level of abstraction which is convenient for understanding the business logic, rather than at a level that is convenient for processing by a platform.

A system must eventually be implemented by deploying its business logic onto one or more platforms. In the MDA approach, this is achieved by converting PIMs into PSMs. This appears to add an additional stage in an MDA development that could be avoided by simply developing for a single platform initially. However, having produced a detailed model of the business logic of a system, its implementation is often a matter of routine translation into the platform of choice. Some or all of this may be automated. If automated transformations are available to deploy the system to a variety of platforms, then developing for all of those platforms is little more expensive than developing for just one.

The MDA approach also provides a principled approach to integrating legacy systems. PIMs for such systems must first be obtained using some combination of reverse engineering and a conventional analysis (an object-oriented analysis if the modeling language is UML). Once modeled, these systems can be integrated into larger confederations as easily as nonlegacy applications developed using the MDA approach.

The implications of the MDA idea are far reaching. With the largest part of its benefit coming from the idea that computing systems should be developed in a language related to the application domain rather than the deployment platform, the approach is effectively raising the level of abstraction at which systems are developed into the realm of domain-specific business languages. Such languages could dramatically increase the productivity of development organizations. Much research and standardization efforts surrounding the MDA initiative is focused on developing domain-specific languages.

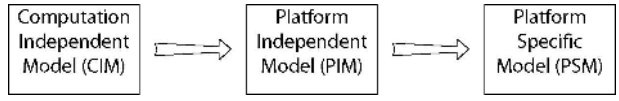
The MDA and SOC are complementary approaches to the problem of maintaining and integrating systems. SOC complements the MDA approach by providing a template for structuring business systems and integration solutions, the service paradigm. The MDA complements SOC by promoting the early modeling of systems to understand their operation, a necessary prerequisite when wrapping a system with a service interface.

Models and Model Transformation

The MDA guide provides more specific details of the MDA approach. In the MDA terminology, a business-domain model is either a Computation Independent Model (CIM) or Platform Independent Model (PIM). CIMs represent the interrelationships of domain concepts and are intended as early analysis and requirements models, to be produced by domain experts but not requiring systems development expertise. PIMs are refined system models that represent a complete specification of the structure and behavior of an application, independently of platform decisions. Models describing the implementation of a system on a particular platform are referred to as Platform Specific Models (PSM). In the MDA approach, the business knowledge for a system is first captured in CIMs and PIMs, then mapped to PSMs for the supported technical infrastructures, as shown in Figure 1.

In the MDA vision, model transformations automate repetitive development tasks, for example, the translation of PIMs into PSMs. Figure 2 illustrates a model transformation, also referred to as a mapping. This conception of model transformation is referred to as *the MDA pattern.* A source model, usually a PIM, is transformed into a target model, usually a PSM.

Figure 1. Models in the MDA approach

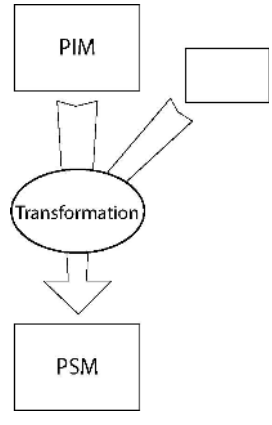


The figure shows the transformation incorporating some additional information. Transformations often require additional guidance, for example, incorporating design decisions to guide exactly how the PIM should be implemented on the platform. This information is usually provided in the form of *markings*, annotations to designs associated with a particular transformation.

Transformations need not be uniquely associated with the transition from PIM to PSM. Transformations can also potentially be used to perform complex edits, such as refactorings. In a situation where multiple CIMs or PIMs use a number of different domain languages, transformations can be used to manage the relationships between these platform-independent models, for example, by combining domain-specific representations into more generic representations. Model transformations can also be used to derive new views of a system from existing specifications, produce documentation, or derive formal analysis models as in Skene and Emmerich (2003).

The OMG is in the process of standardizing languages for the description of transformations (OMG, April 2002). Transformations will be described in terms of *model type mappings* in which rules will specify how typed elements in the source models are transformed into elements in the target models. Also possible are *instance mappings* in which model elements are marked by the user as the source for a transformation pattern, regardless of their underlying type. It is expected that most mappings will allow a combination of these approaches with model type mappings managing routine corre-

Figure 2. A pattern for model transformations



spondences between source and target and instance mappings allowing the user to direct model transformation.

The Unified Modeling Language (UML)

UML (OMG, January 2003) is an object-oriented graphical language that has been widely adopted in industry to represent software designs, particularly those which are to be implemented in one of the currently dominant object-oriented programming languages, such as Java, C++, and C#. However, it is its heritage as an analysis language (in the sense of requirements capture and problem-domain modeling) that makes it a suitable core representation for the MDA approach, as the ability to represent a technology-neutral PIM is provided by the facilities that support the modeling of problem domains in object-oriented analysis.

In this chapter, we describe how UML can be used as a basis for modeling electronic services. This modeling depends on domain-specific extensions to UML, delivered using profiles. Profiles are a UML extension mechanism whereby the innate notations provided by the UML can be augmented with labels, called *stereotypes,* tagged values and constraints which provide semantic refinement, annotations, and syntactic refinement, respectivly.

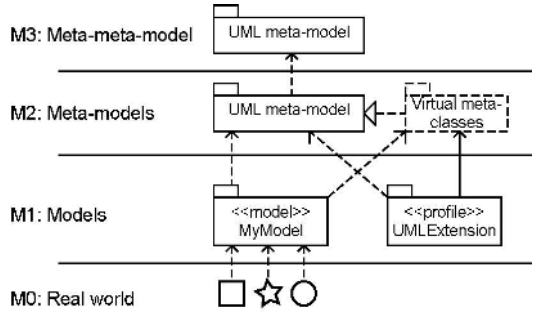
Extensions to the UML provide the means to denote designs more concisely. By introducing domain-specific vocabulary using profiles, the designer can avoid the overhead of expressing standard aspects of the design in fine detail. Extending the UML is a useful activity when following an MDA development process. In the MDA models are specific to unique, well-encapsulated semantic domains, for example, a PIM for a particular type of business process or a PSM targeting a specific platform. It is convenient to describe designs in these domains using the vocabulary of the domain.

UML is based on a conceptual architecture that is divided into four meta-modeling layers as shown in Figure 3. The lowest level is the data layer (M0), in which objects such as data-patterns in computer memory and other real-world phenomena, including people and things, are supposed to reside. The elements in the lowest level are classified by types in the UML models that analysts and designers produce, which hence reside at the next meta-level (M1). UML model elements are, in turn, objects of classes in the UML meta-model (M2). Attached to these meta-classes are semantic descriptions and syntactic constraints that control the meaning and applicability of the UML. The meta-model at level M2 is self-describing so can also be regarded as residing in level M3 (and plausibly all higher levels).

Profiles then, are a means of refining classes, semantics and syntactic constraints at the M2 level. Confusingly, profiles are defined at the M1 level so that they can be denoted using UML and deployed by including them with any UML model that requires their language extensions. They can therefore be regarded as injecting *virtual meta-classes* into the UML meta-model (M2).

When presenting profiles, it is common to first present a domain model (Frankel, 2003). Domain models directly describe the semantic domain, independently of the need to

Figure 3. Meta-modeling architecture of the UML



refine the semantics of the UML meta-model. Domain models are usually UML class models.

In the context of the MDA, domain models have additional significance. MDA transformations that proceed from designs annotated with profile elements must be informed not only by the design model but by the semantics of the extensions used in this model. This makes the provision of domain models as a reference for UML extensions vital when supporting the MDA development approach. When presenting our profile for electronic service systems below, we first present our domain model. This introduces the concepts and relationships intrinsic to electronic services and serves as the semantic basis for our profile. This was also the approach taken by the OMG when standardizing the EDOC profile described in the next section.

Profile for Enterprise Distributed Object Computing (EDOC)

The EDOC profile is an OMG standard, intended to simplify the development of open distributed systems when taking a UML modeling approach (OMG, May 2002). We present a brief overview of EDOC as an example of the support for developing electronic services already provided by the UML and MDA initiatives.

The EDOC profile actually consists of several related profile specifications supporting an MDA development approach. The Enterprise Collaboration Architecture (ECA) profiles enable the definition of an EDOC system in a platform-independent manner by recursive decomposition into collaborating components. Technology-specific profiles provide language extensions supporting common implementation platforms for EDOC systems, in particular the Enterprise Java Beans (EJB) platform (Sun Microsystems, 2001). According to the MDA process, designs are first refined using the ECA profile then deployed using a particular platform technology. The EDOC profiles are:

- **The Enterprise Collaboration Architecture (ECA) Profile:** Comprised of:
  - *The Component Collaboration Architecture:* Allows the specification of the system as a set of collaborating components. Component processes and collaboration protocols (choreography) are modeled in a manner compatible with the ebXML process language specification (Gibb & Damodaran, 2002).
  - *Business Process Profile:* Specializes CCA to model systems in the context of the enterprise that they support. Includes extensions to indicate dependencies between business processes and associations between business tasks and the roles that perform them.
  - *Entities Profile:* Enables the description of concepts in the problem domain of the system, in particular the data-types of entities and their relationships. Entities are defined as CCA components, integrating data and process views.
  - *Events Profile:* Extensions to the CCA to model event-driven systems.
  - *Relationships Profile:* More rigorous concepts of relationships for business and software entities than UML provides by default.
- **The Patterns Profile:** Increases the expressive power of the ECA by allowing the definition of generic patterns of business object collaborations and the reuse of these patterns in specific bindings.
- **The EJB Profile:** A platform-specific extension allowing the concise modeling of EJB designs. The EJB profile is included in the EDOC profile as an example of the platform-specific modeling support required to develop open distributed systems. Other platform-specific profiles such as the CORBA profile (OMG, April 2002) are also compatible with the general MDA approach.

The EDOC profile reuses many of the concepts introduced in the ISO standard Reference Model for Open Distributed Computing (RM-ODP). RM-ODP standardizes a conceptual model of open distributed systems, supporting the definition of five viewpoint languages that allow the specification of particular distributed system designs. Each viewpoint is tailored to a particular set of concerns for the system. The EDOC profile reuses these viewpoints and redistributes the expressive capabilities of the viewpoint languages into the profiles described above. This unification of the RM-ODP concepts with the UML significantly increases the usefulness of both specifications, lending the former a standard language for representing designs and the latter a rich semantic for describing EDOC systems. The viewpoints are:

- **Enterprise specification:** Models the structure and behavior of a system in the context of the business of which it forms a part. Supported by the CCA and Business Process profiles which model the system and its environment as interoperating components.

- **Computational specification:** Models the implementation of enterprise components. The CCA supports a smooth transition from the enterprise specification to computational specification through recursive decomposition of components.
- **Information specification:** Describes the data environment of the system. Supported by the Entities profile.
- **Engineering specification:** Describes the middleware and services infrastructure of the system. The RM-ODP engineering language allows explicit descriptions of protocol elements and standard services, such as naming and transaction services. EDOC relegates these details to the technology mappings, relying on the platform semantics to provide the infrastructure specification.
- **Technology specification:** Describes the deployment of the system. Supported by technology mappings and the UML's native deployment models.

A complete specification of an EDOC system includes models providing each of these viewpoints. This includes a platform-independent description of the system using the CCA and one or more platform-specific implementations of the system, expressed using technology-specific profiles. Hence, the RM-ODP viewpoint system can be seen to be aligned with an MDA development approach.

Domain Model for Electronic Service Systems

The remainder of this chapter presents a profile for modeling Electronic Service Systems (ESSs) using UML and provides an example of its application for modeling services in the freight domain. The example is based on previous research in the freight domain (Linketscher & Child, 2001), and we will use it to illustrate aspects of electronic services in the following discussion.

As discussed previously in the section on UML, the semantics of profiles are often defined with reference to a domain model. The domain model is a UML model with accompanying natural language descriptions that presents the concepts, entities, and relationships present in semantic domain of the extension. The next subsection describes the sources of concepts from the model and its overall section. Subsequent sections elaborate various aspects of the model.

Analysis

In this section, we discuss the sources of concepts in the domain model for ESSs.

The model is based, in part, on the author's experience with commercial platforms for electronic services. Electronic Service Management Systems (ESMSs) are applications

based on the electronic service paradigm, providing facilities to integrate, control, and enact electronic services. Such systems present service-oriented viewpoints of the enterprise to management, supporting the structuring of businesses as systems of electronic services, and may include data, resource, and process management facilities.

We have considered ESMSs from vendors, including BEA, IBM, Microsoft, Oracle, and HP (see references for all). While terminology and emphasis vary across the range of platforms, the proposed model captures the essential common concepts and technology elements: the modularization pattern for capabilities and services is reflected in the design and management tools included in the platforms. Databases, electronic service management systems, and, in some cases, workflow engines, constitute the core technology for all the platforms.

Alignment between these platforms is supported by joint standardization efforts promoted by the respective vendors. An exemplary case in terms of the central role of workflow is BPEL (Business Process Execution Language) (Andrews et al., 2003). The model proposed takes into consideration existing standards, as well as evolutionary trends for standard frameworks, both in the commercial and scientific domain exemplified by the activities of W3C, OASIS, and the Global Grid Forum.

In addition, the model reflects concepts inherent to established development methodologies for electronic services adopted within the industry (McCarthy) and published methodologies associated with the platform specifications. For example, the HP Service Composer structures its usage around a methodology for the definition and development of electronic services. The essence of the methodology is captured in the following steps. We identify the corresponding elements of our model in parentheses:

1. **Define Public Business Processes:** The developer defines the public workflow that clients will use to interact with the service. The developer either selects an existing process definition or defines new ones (ElectronicService, ServiceOffer, Capability).
2. **Program Web Service Interfaces:** The developer generates the Web Services Description Language (WSDL) files, which describe the Web services associated to the process of step one.
3. **Generate Business Objects and Data:** The developer generates or creates connections to the business objects and data that support the service (Capability, InformationItem, BusinessEntity, ITSystem).
4. **Define Internal Business Processes:** The developer defines the internal workflow specifying the operational logic for the service. For pre-existing workflows, the developer builds access points to relevant process nodes (Capability).
5. **Map Public Interfaces:** The public interfaces defined in steps one and two are mapped to backend logic from steps three and four. As an example, a WSDL interface might be mapped to a backend component for its concrete implementation (Capability, CapabilityRole, RoleAssignment).
6. **Package the Service:** The various components and descriptor files that make up the service are combined into a deployment unit. The deployment unit can vary depending on the target platforms (ElectronicService).

7. **Deploy the Service:** The service is deployed onto the various components of the runtime platform (for example, application server, workflow engine, ERP — enterprise resource panning—system, Web service infrastructure) (ITSystem, RoleAssignment).
8. **Advertise the Services:** Once a service has been deployed, it can be offered to clients. For example, entries for the related Web services can be added to a UDDI repository.
9. **Monitor Running Services:** Graphical tools should provide an end-to-end view of the service at instance level or as aggregates (ESMS).

Aligned with industry trends such as ebXML (Gibb & Damodaran, 2002) and technology trends, such as Web services, HPSC is representative of the state of the art in commercial systems. The conceptual framework employed by the HPSC is reflected in the concepts of business services, electronic services, and ESMSs presented in the domain model.

Elements in the model fall into two categories: those representing the abstract concepts inherent to electronic services and those representing the IT infrastructure used to implement such systems. In the following subsections, we describe the domain model for electronic services in more detail.

We begin by establishing a working definition of electronic services. This definition is abstracted from the technical details of electronic service provision, including decisions regarding implementation technologies or supporting platforms.

Having established the features of electronic services, we consider their concrete implementation in a business setting. Business capabilities require their roles to be fulfilled by business entities, which include electronic services, staff, resources, external services, and clients.

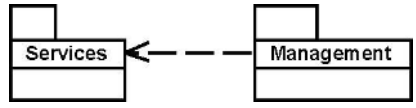
Finally, we introduce a model of infrastructure components, reflecting the common practice of employing commercial components such as databases, electronic service management systems, and workflow engines when implementing ESSs.

The domain model is divided into two packages as shown in Figure 4. These partition the elements pertaining to services from those which represent the IT infrastructure for managing services.

Electronic Services

The notion of a *business service* enables the management within an enterprise of *capabilities* to deliver some benefit to a consumer. The term *capability* refers to the coordination of simpler tasks to achieve an end; the concept is used to raise the level of abstraction when describing the way that a business behaves. A capability can be associated with a workflow specification to show how its end is achieved. When describing business services, capabilities are divided into those involved in *provisioning* the service and those providing the *content* of the service.

Figure 4. Subpackages within the ESS domain model



The content of a service is the set of capabilities that deliver the benefit of the service to the client. For example, the content of a freight service refers to the capability of moving goods from one place to the other.

Provisioning refers to the business channel between the provider and the consumer of a service. In a freight service, provisioning covers selection, product offer, pricing, and interaction processes that the freight company applies to its customers.

Content and provisioning are complementary aspects of a service: provisioning logic depends on the capabilities that the provider can support. The capabilities made available to consumers depend on the provisioning logic adopted by the provider. In the example, the option of delivery tracking might be made available only to selected customers.

Because business services require communication between the provider, the consumer, and entities fulfilling other supporting roles, it is natural to provide interfaces to business services using communication technologies, such as computer networks, and supporting software, such as middleware for distributed systems. An electronic service is a set of meta-data, communication interfaces, software, and hardware supporting a business service (Marton, Piccinelli & Turfin, 1999).

Middleware services and computing resources also provide the opportunity to implement new business services with highly automated content, and this is an expected benefit of the electronic service model. However, despite the similarities, the notion of electronic services should not be confused with middleware services. Electronic services are associated with business capabilities, and this association is significant to the way in which electronic services are used and coordinated. The workflow descriptions associated with a service's capabilities can be used to coordinate the service: internally, to marshal the involved capabilities and resources and establish the relationship between content and provisioning, and externally, to manage the interaction between the service and its clients and environment.

Figure 5 shows the part of the services domain model related to the composition of capabilities into services. The elements shown are now described:

- **ElectronicService:** The encapsulation or realization of a business service using electronic interfaces. Electronic services have any number of provisioning capabilities and a single top-level content capability (the capability to deliver the service). Services can be composed of subservices, in which case the content capability coordinates the content of each subservice, and each subservice must

have a provisioning capability that makes a service offer to a role in the coordinating content capability.

- **Capability:** A behavior which realizes some benefit to the business, described by a workflow. A number of roles perform actions and cooperate to complete some task. Capabilities can be composed in a hierarchy. The workflow of a coordinating capability constrains the order of tasks in the component capabilities.
- **CapabilityRole:** A capability role identifies the behavior of a worker or resource in a coordinated task. Capability roles can be assigned to actual business entities as discussed shortly.
- **InformationItem:** An identifier for a piece of information about an enterprise that is relevant to a task. Some workflow actions require information as a prerequisite and produce or process information as by-product of their enactment.
- **Observation:** Observations infer new information from existing information. This captures the idea that not all derived information is produced by a particular action. When the condition of the observation is satisfied, new information may be introduced by the observation expression.

Figure 5. Capabilities view of the services domain model

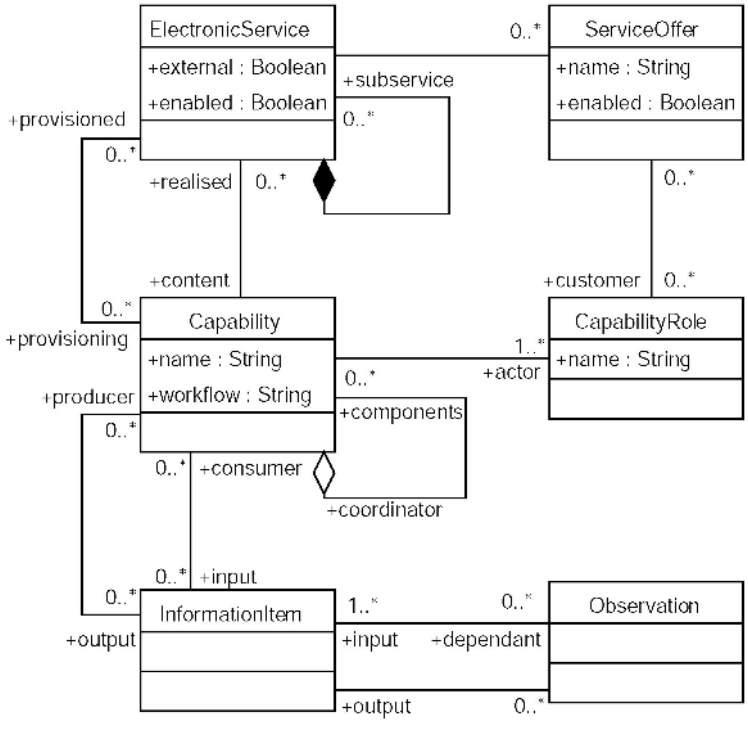
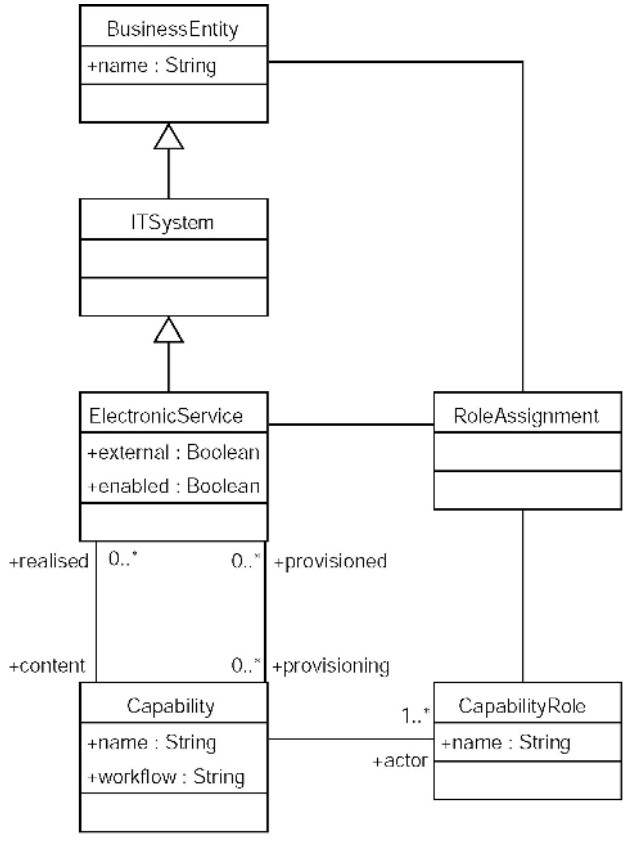


Figure 6. Implementation view of the services domain model



Business Entities

Complementary to the abstract view of business capabilities are models of the concrete assets in an enterprise and their assignment to capability roles to realize a service. Electronic services are business assets themselves. Services cooperate at a peer-to-peer level by fulfilling roles in capabilities. Figure 6 shows the domain model classes.

- **BusinessEntity:** A business entity is a person, resource, or system that can fulfil one or more roles in a capability.
- **Service Offer:** A service offer is made to a capability role (typically that of the 'customer'). That capability role must be associated with one of the provisioning mechanisms of the service.
- **RoleAssignment:** Captures the idea that business entities can be assigned to perform roles in capabilities on behalf of a particular service (capabilities may be employed by multiple services, so it is necessary to state what service an entity is

assisting). Deployment models, including role assignments, represent a snapshot of the enterprise with a particular disposition of resources.

- **ITSystem:** An IT system is a computing system that can perform a role in a capability. Electronic services are intended to provide integration and automated coordination. This class allows the identification of the components providing these services, possibly as a prelude to an MDA-style development activity. The subsequent subsection, Electronic Service Management Systems, provides refinements of this stereotype to identify likely management applications.

Additional classes not shown in Figures 5 and 6 are now discussed:

- **Property and HasProperties:** Properties capture different types of meta-data about capabilities. Such meta-information mainly refers to functional and nonfunctional requirements for a capability. For example, a property for a negotiation capability is to be usable only with a certain type of customers. The following classes inherit from HasProperties to enable the attachment of properties: BusinessEntity, CapabilityRole, Capability and Service. The properties mechanism maps onto the tagged-value mechanism in UML in the profile definition.
- **Group and Groupable:** Experience with the HP Service Composer revealed the benefit of composing capabilities into higher-level aggregates called *clusters,* in which capabilities exhibited functional overlaps, dependencies, mutual ownership, or other subjective similarities. There is also often the need to group services into related offerings or *service packs.* Group and Groupable provide a single mechanism for hierarchical grouping. The following elements inherit from Groupable and, hence, may appear in a Group: CapabilityRole, Capability, BusinessEntity, InformationItem, Service and Group. Grouping is implemented by UML's package mechanism in the profile definition.

Electronic Service Management Systems

In this section, we introduce a preliminary model of IT infrastructure components involved in the management and implementation of ESSs. Because it is common practice to employ such components when structuring business capabilities as electronic services, we believe that a pragmatic modeling approach for ESSs must represent these components explicitly. The following discussion identifies several common types of infrastructure component.

Our domain model describes abstractly the effect that activities have on the information in their environment, for example, the known locations of resource or statistics, such as the total revenue for a service. Such information can have a role in coordinating capabilities and may be maintained and leveraged using databases or other accounting mechanisms.

Business services encapsulated by electronic services benefit from additional communication and provisioning channels. However, the notion of electronic service includes workflow descriptions for the capabilities from which services are composed. This suggests that services may be automatically coordinated or enacted using commercial workflow engines.

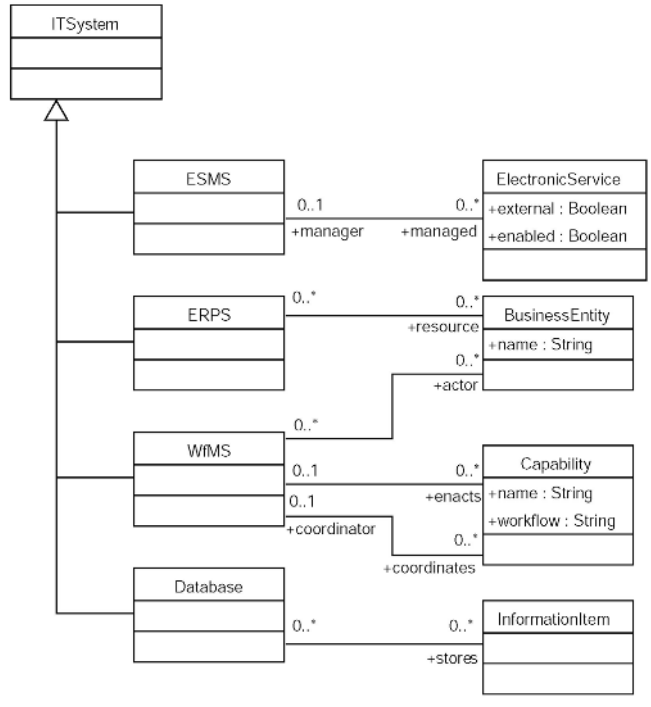
There may also be a need to manage the resources required by a service, support for which is provided by Enterprise Resource Planning (ERP) applications.

Generally, if electronic services are in place, there will be the need and opportunity to integrate them using a technical infrastructure. This integration is the key benefit of the SOC paradigm. ESMSs bundle facilities for data and workflow management with development tools based on the service paradigm.

The management domain model shown in Figure 7 identifies several common types of management components and their relationship to electronic services.

- **ESMS:** An application offering an enterprise-oriented management view of an electronic service environment. For example, the HP Service Composer or the DySCo research prototypes (Piccinelli & Mokrushin, 2001). Other candidate technologies might be an application service offering a middle-tier of business logic with a web server providing the management interfaces.

Figure 7. Management domain model



- **WfMS:** A workflow management system, either embodying a capability (enactment) or coordinating a number of subcapabilities. Examples of workflow applications are IBM's MQ-Series Workflow and PeopleSoft's PeopleTools and Integration Broker.
- **ERPS:** An Enterprise Resource Planning System, dedicated to coordinating entities in the system, presumably making them available to fulfil capability roles. We do not consider resource planning in this paper, although it interacts at a functional level with coordination based on capabilities, and future work may provide a combined modeling approach. Examples of ERP systems are SAP's mySAP and Baan's iBaan.
- **Database:** Most enterprises use databases to store information about the enterprise. Establishing a relationship between the (conceptual) information items and the databases that store them allows a modeler to check whether the information required by a business entity to fulfil a capability role is available in its context. Popular databases are Oracle and MySQL.

Electronic Service Systems Profiles

The following tables relate elements in the domain model to profile elements and elements in the UML domain model. This style of profile definition is compatible with both UML version 1.5 (OMG, January 2003) and UML 2 (OMG, September 2003).

All name attributes in the domain models map to the name attribute of the class element in the UML meta-model. All associations in the domain model map to associations in models. Stereotypes on AssociationEnds are used to disambiguate associations where more than one exists between the same two domain model elements.

Profile Usage Example

We now present an example of the profile in use to model a freight moving service, based on previous research in the freight domain (Linketscher & Child, 2001). This example was also used to demonstrate the HP Service Composer.

We present three diagrams corresponding to the three views supported by our profile: an abstract view of a service and its underlying capabilities; a snapshot of the enterprise with business entities implementing roles in capabilities; and a view of the IT infrastructure supporting services.

Figure 8 shows the freight service and the capabilities that support it. The service is provisioned by a tendering capability. This service bids in a reverse auction. Simulta-

Table 1. Stereotypes in the ESS profile

Domain model element	Stereotype	UML base class	Parent	Tags
ElectronicService	Service	Class	--	external enabled
ElectronicService.content	content	AssociationEnd	--	--
ElectronicService.provisioning	provisioning	AssociationEnd	--	--
ElectronicService.component	component	AssociationEnd	--	--
Capability	Capability	Class	--	--
Capability.input	input	AssociationEnd	--	--
Capability.output	output	AssociationEnd	--	--
CapabilityRole	CapabilityRole	Class	--	--
InformationItem	InformationItem	Class	--	--
Observation	Observation	Class	--	condition observation
Observation.input	input	AssociationEnd	--	--
Observation.output	output	AssociationEnd	--	--
BusinessEntity	BusinessEntity	Class	--	--
ServiceOffer	ServiceOffer	Class	--	enabled
ServiceImplementation	Fulfills	Association	--	service
ITSystem	ITSystem	Class	Business- Entity	--
ESMS	ESMS	Class	ITSystem	--
WfMS.actor	wfactor	AssociationEnd	--	--
WfMS.enacts	enacts	Class	--	--
WfMS.coordinated	coordinates	Class	--	--
ERPS	ERPS	Class	ITSystem	--
Database	Database	Class	ITSystem	--

neously it coordinates resources required for the freight movement. Resources are traded in an online market to drive down the overhead of the transport. A successful tender resorts in a move order. This is an example of information in the ESS.

Each capability has an associated workflow description. The workflow for the handover capability is shown using an informal notation. The role names present correspond to associated capability roles.

Table 2. Tags in the ESS profile

Domain model element	Tag	Stereotype	Type	Multiplicity
Service.external	external	Service	Boolean	0..1
Service.enabled	enabled	Service	Boolean	0..1
Capability.workflow	workflow	Capability	String	0..1
Observation.condition	condition	Observation	String	1
Observation.observation	observation	Observation	String	1
ServiceImplementation.service	service	Fulfills	Class	1
BusinessEntity.external	external	BusinessEntity	Boolean	0..1
ServiceOffer.enabled	enabled	ServiceOffer	Boolean	0..1

Figure 8. Services and capabilities in the Freight mover example

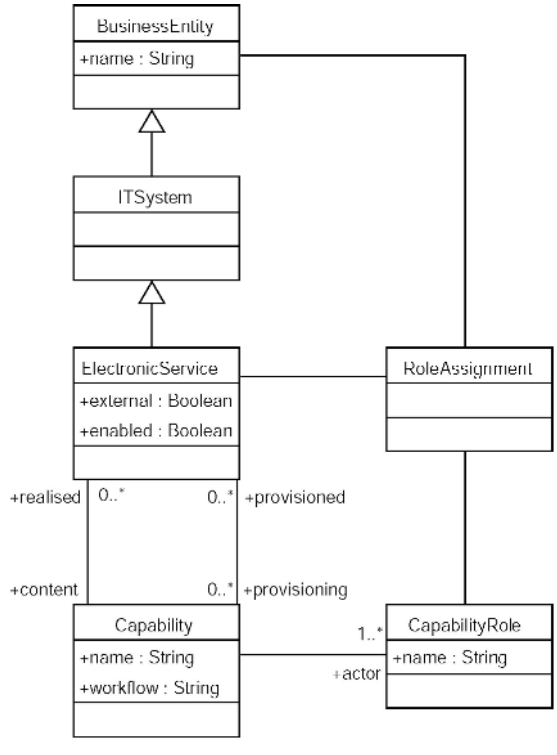


Figure 9 shows a deployment view of the service in operation. The model shows the association of commercial entities with the roles associated with the handover capability.

Finally, Figure 10 shows a view of the IT infrastructure supporting the freight service. The overall service is managed by an ESMS. The tender and auction capabilities are

Figure 9. Resource assignments in the Freight mover example.

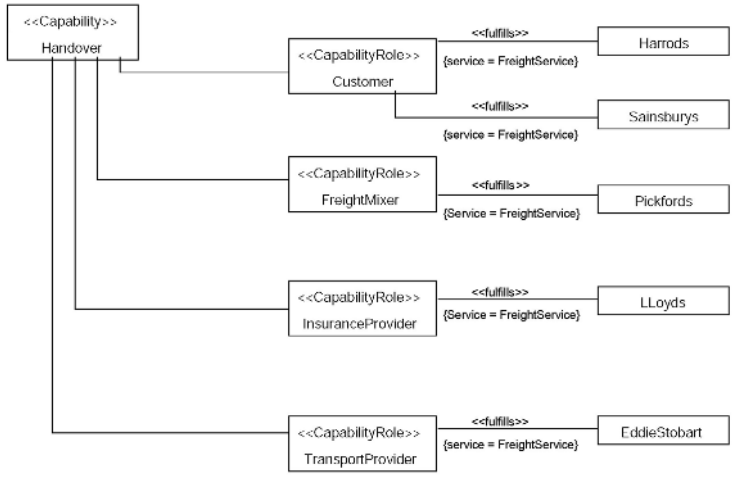
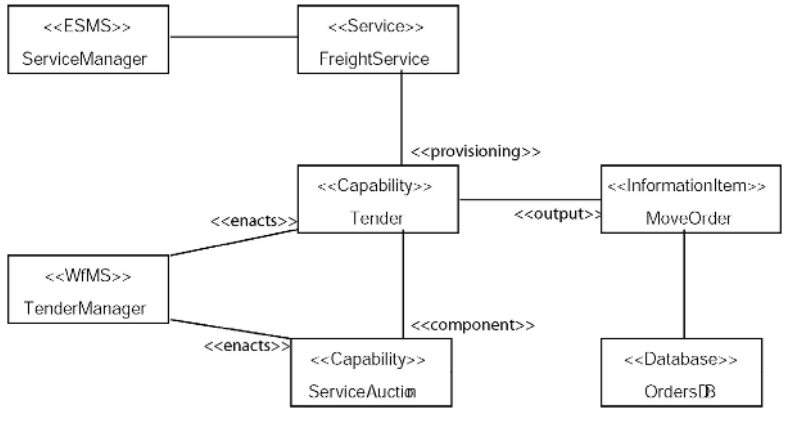


Figure 10. Infrastructure components in the Freight mover example.



enacted by a workflow engine. An orders database manages information produced by successful tenders to move goods.

Conclusion and Future Trends

Service orientation and model-driven architectures closely represent complementary approaches to tackling the complexity of software systems engineering. On the one hand,

systems are decomposed into functional units (services) that can be composed in a modular and flexible way. On the other hand, the engineering of individual services and of complete solutions is based on a coherent management of multiple views (models) of the underlying system. The combination of service orientation and model-driven architectures results in a comprehensive conceptual framework for systems engineering.

In this chapter, we have proposed a concrete example of the possible synergies between service orientation and model-driven architecture. Using the extension capabilities of UML, the conceptual framework defined for ESSs has been used to enhance standard UML modeling tools. The resulting modeling and development environment intrinsically supports the realization of service-oriented systems.

Models of systems developed using a profile of this kind provide a high-level view of a system in terms of the concepts underlying its design, in this case, the electronic service paradigm. The benefit of such a view is in understanding and documenting the system and serving as a starting point for refinement based development, as suggested by the MDA development approach. The profile identifies elements, such as services and workflow specifications, pertinent to platform artifacts, and we suggest that such models could therefore serve as high-level PIMs appropriate as the source for MDA transformations to platform-specific representations. Given the high level of abstraction of the models and the great variety of possible implementation platforms and strategies for electronic service systems, we are currently content to leave this as a manual transformation, the details of which should be determined for each particular development. However, if the increasing popularity of the MDA results in the emergence and widespread adoption of standards and tools supporting automated transformations, it may also be profitable to formally define transformations to a variety of platforms.

References

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., et al. (2003, May 5). Business process execution language for web services version 1.1. Retrieved August 9, 2004, from *http://www.ibm.com/developerworks/library/ws-bpel/*

Baan. iBaan. Retrieved August 9, 2004, from *http://www.baan.com/*

BEA. Web logic integration. Retrieved August 9, 2004, from *http://www.bea.com/content/news_events/white_papers/BEA_WL_Integration_ds.pdf*

Frankel, D. S. (2003). *Model driven architecture: Applying MDA to enterprise computing.* John Wiley & Sons.

Gibb, B., & Damodaran, S. (2002). *ebXML: Concepts and application.*

Global Grid Forum. Retrieved August 9, 2004, from *http://www.ggf/org/*

HP. (2002). HP service composer user guide.

IBM. Websphere MQ workflow. Retrieved August 9, 2004, from *http://www.ibm.com/software/integration/wmqwf/*

IBM. WebSphere Studio Application Developer. Retrieved August 9, 2004, from *http://www.ibm.com/websphere*

ISO/IEC, ITU-T. Open distributed processing — Reference model — Part 2: Foundations. (ISO/IEC 10746-2, ITU-T Recommendation X.902).

Linketscher, N., & Child, M. (2001). *Trust issues and user reactions to e-services and e-marketplaces: A customer survey.* DEXA Workshop on e-negotiation.

MacDonald, M. (2003). *Microsoft .NET distributed applications: Integrating XML Web services and .Net remoting.* Microsoft Press.

Marton, A., Piccinelli, G., & Turfin, C. (1999). *Service provision and composition in virtual business communities.* IEEE-IRDS Workshop on Electronic Commerce.

McCarthy, J. A design center for Web services. Retrieved August 9, 2004, from *http://www.webservices.org*

MySQL AB. MySQL database. Retrieved August 9, 2004, from *http://www.mysql.com/*

OASIS. Retrieved August 9, 2004, from *http://www.oasis-open.org/committees/*

Object Management Group (OMG). (2002, April). MOF 2.0 queries/views/transformations RFP. (Document — ad/02-04-10).

Object Management Group (OMG). (2002, April). UML profile for CORBA specification. (Document — formal/02-04-01).

Object Management Group (OMG). (2002, May). UML profile for enterprise distributed object computing specification. (Document — ptc/02-02-05).

Object Management Group (OMG). (2003, January). Unified Modeling Language (UML), version 1.5. (Document — formal/03-03-01).

Object Management Group (OMG). (2003, June). MDA guide version 1.0.1. (Document — omg/03-06-01).

Object Management Group (OMG). (2003, September). UML 2.0 infrastructure final adopted specification. (Document — ptc/03-09-15).

Oracle. Oracle database products. Retrieved August 9, 2004, from *http://www.oracle.com*

Piccinelli, G., & Mokrushin, L. (2001). *Dynamic e-service composition in DySCo.* Workshop on Distributed Dynamic Multiservice Architecture, IEEE ICDCS-21.

SAP. mySAP. Retrieved August 9, 2004, from *http://www.sap.com/*

Skene, J., & Emmerich, W. (2003, October). *A model-driven approach to non-functional analysis of software architectures.* Proceedings of the18th IEEE Conference on Automated Software Engineering.

Sun Microsystems. (2001). Enterprise Java-Beans (EJB) specification v2.0. Retrieved August 9, 2004, from *http://java.sun.com/products/ejb/docs.html*

W3C. Web services activity. Retrieved August 9, 2004, from *http://www.w3.org/2002/ws*

Chapter VII

Service-Oriented Enterprise Architecture

Maarten W.A. Steen
Telematica Institute, The Netherlands

Patrick Strating
Telematica Institute, The Netherlands

Marc M. Lankhorst
Telematica Institute, The Netherlands

Hugo W.L. ter Doest
Telematica Institute, The Netherlands

Maria-Eugenia Iacob
Telematica Institute, The Netherlands

Abstract

Service orientation is a new paradigm, not only for software engineering but also for the broader topic of enterprise architecture. This chapter studies the relevance and impact of the service concept and service orientation to the discipline of enterprise architecture. It provides ideas on how to set up a service-oriented enterprise architecture. It is argued that a service-oriented approach to enterprise architecture provides better handles for architectural alignment and business and IT alignment, in particular.

Introduction

Continuing globalization, the economic downturn, mergers and acquisitions, and changing customer demands are forcing enterprises to rethink and restructure their business models and organizational structures. New products and services need to be developed

and delivered better, faster, and cheaper due to increasing international competition. Therefore, enterprises have to be increasingly efficient, flexible, and innovative to be successful. They will focus more on core competencies and outsource other activities to dynamically selected partners to deliver the best possible customer value and the shortest time-to-market.

In order to manage all these changes and stay competitive, enterprises have started to develop enterprise architectures. These bring together all architectures modeling specific aspects of an enterprise. They provide a way for managers and enterprise architects to assess the impact of changes in one aspect of the enterprise's operations on the other aspects.

The emergence of the service-oriented computing (SOC) paradigm and Web services technology, in particular, has aroused enormous interest in service-oriented architecture (SOA). Probably because such hype has been created around it, there are a lot of misconceptions about what SOA really is. Numerous Web services evangelists make us believe that if you would divide the world into service requestors, service providers and a service registry, you would have a service-oriented architecture (for example, Ferris & Farrell, 2003). Others emphasize that SOA is a way to achieve interoperability between distributed and heterogeneous software components, a platform for distributed computing (for example, Stevens, 2002). The interesting thing is that the service concept applies equally well to the business as it does to software applications. Services provide the *units of business* that represent value propositions within a value chain or within business processes. Even though dynamic discovery and interoperability are important benefits of Web services, a purely technological focus would be too limited and would fail to appreciate the value of the (much more general) service concept. SOA represents a set of design principles that enable units of functionality to be provided and consumed as services. This essentially simple concept can and should be used not just in software engineering but also at all other levels of the enterprise architecture to achieve ultimate flexibility in business and IT design.

The main objective of this chapter is to study the relevance and impact of the service concept and service orientation on the discipline of enterprise architecture. The chapter answers the following questions:

- What is enterprise architecture and why is it important?
- What is the current state of practice in enterprise architecture?
- Why should enterprises consider moving to a service-oriented enterprise architecture?
- What are the implications of service orientation for enterprise architecture?
- What support is required for doing service-oriented enterprise architecture?
- What road maps exist for moving to a service-oriented enterprise architecture?

The rest of the chapter is structured as follows. First, we survey the state of the art in enterprise architecture and architectural support. Then we study the relevance and

impact of the service concept and service orientation on the discipline of enterprise architecture. This is followed by a number of emerging trends and adoption strategies. We conclude with issues for further research.

Enterprise Architecture

Enterprise architecture has the effective purpose to align the strategies of enterprises with their business processes and their (business and IT) resources (Wegmann, 2003; Zachman, 1987). An enterprise architecture for an organization combines and relates all architectures describing particular aspects of that organization. META Group, for example, defines the enterprise architecture to consist of the enterprise business architecture, the enterprise information architecture, the enterprise-wide technology architecture, and the enterprise application portfolio (Buchanan & Soley, 2002). The goal of enterprise architecture is to provide *insight* in the organizational structures, processes, and technology that make up the enterprise, highlighting opportunities for efficiency improvements and improved alignment with business goals.

Enterprise architecture is important because organizations need to adapt increasingly fast to changing customer requirements and business goals. This need influences the entire chain of activities of an enterprise from business processes to IT support. Moreover, a change in one architecture may influence other architectures. For example, when a new product is introduced, someone should be made responsible for it, and business processes for production, sales, and after-sales need to be adapted. It might be necessary to change applications or even adapt the IT infrastructure. To keep the enterprise architecture coherent, the relations between the different types of architecture must be clear, and a change should be carried through methodically in all architectures.

Architectural alignment and business and IT alignment, in particular, have proved to be difficult problems in enterprise architecture. On the one hand, this is due to differences in architectural modeling methods. Business analysts are capable of modeling complex business processes. Likewise, IT architects are capable of designing complex applications. Unfortunately, the two cannot understand each other's designs because they do not have a common vocabulary and language. On the other hand, there is no overarching set of design rules governing the structuring of the various architectures making up the enterprise architecture. In practice, each type of architecture is supplemented with guidelines and best practices for optimal design. The use of such *design principles* is well known in software engineering, but also in the field of business process modeling a number of guidelines have been assembled in various methods. For example, Biemans et al. (2001) describe guidelines, such as "use domain-specific terminology, notation, and conventions," and "use a limited number of pre-defined abstraction levels; the choice of abstraction levels should be an opportunistic, domain-specific one". These principles render optimal architectures that, however, may constitute an enterprise architecture that is not optimal or even aligned. Therefore, one also wishes for some form of enterprise-wide design optimization. The well-known practical approach is to generate views or mappings of one architecture onto another and analyze the result for possible discrep-

ancies. A simple example is to assign actors to process activities and analyze whether the result makes sense from the actor's point of view. In this way, architectures can be pair-wise aligned. Until now, enterprise architecture lacks concepts for expressing global optimization and criteria that guide this optimization across different architectures. We return to these problems later. First, we survey some of the available frameworks, methods, and modeling techniques for enterprise architecture.

Frameworks

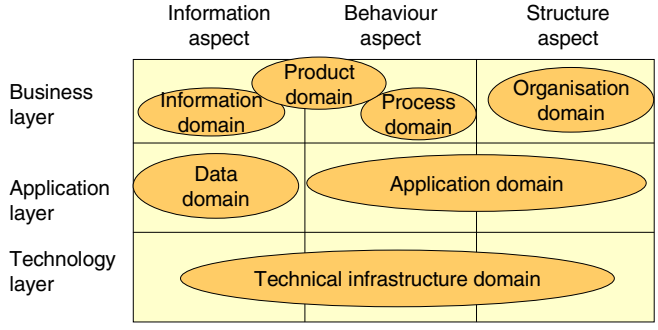
In order to define the field and determine the scope of enterprise architecture, both researchers and practitioners have produced a number of *architecture frameworks.* Frameworks provide structure to the architectural descriptions by identifying and sometimes relating different architectural domains and the modeling techniques associated with them. Well-known examples of architectural frameworks are:

- Zachman's "framework for enterprise architecture" (see Sowa & Zachman, 1992; Zachman, 1987). The Zachman framework is widely known and used. The framework is a logical structure for classifying and organizing the representations of an enterprise architecture that are significant to its stakeholders. It identifies 36 views on architecture (cells), based on six levels (scope, enterprise, logical system, technology, detailed representations, and functioning enterprise) and six aspects (data, function, network, people, time, and motivation).
- The Reference Model for Open Distributed Processing (RM-ODP) is an ISO/ITU Standard (ITU, 1996) which defines a framework for architecture specification of large distributed systems. It identifies five viewpoints on a system and its environment: enterprise, information, computation, engineering, and technology.
- The architectural framework of The Open Group (TOGAF) is completely incorporated in the TOGAF methodology (http://www.opengroup.org/architecture/togaf8/index8.htm). TOGAF has four main components, one of which is a high-level framework defining three views: Business Architecture, Information System Architecture, and Technology Architecture.

For the remainder of this chapter, we will use a very simple framework (Jonkers et al., 2003) to illustrate our ideas, which is based on the frameworks mentioned above. This framework, which is illustrated in Figure 1, uses just three layers and three aspects. The layers – *business*, *application,* and *technology* – roughly correspond to the enterprise, logical system, and technology levels in the Zachman framework. The aspects – *structure*, *behavior,* and *information* – correspond to the network, function, and data aspects in the Zachman framework.

As shown in the figure, different known conceptual domains can be projected onto this framework. Frameworks like this provide clues as to which domains may be relevant for modeling and analyzing but do not provide guidelines for relating and aligning different

Figure 1. Architectural framework



architectural domains, nor for optimizing architecture domains or the entire set of domains.

Methodology

Most of the architecture frameworks are quite precise in establishing what elements should be part of an enterprise architecture. However, to keep the enterprise architecture coherent during its life cycle, the adoption of a certain framework is not sufficient. The relations between the different types of domains, views, or layers of the architecture must remain clear, and any change should be carried through methodically in all of them. For this purpose, a number of methods are available, which assist architects through all phases of the life cycle of architectures.

An architecture method is a structured collection of techniques and process steps for creating and maintaining an enterprise architecture. Methods typically specify the various phases of an architecture's life cycle, what deliverables should be produced at each stage, and how they are verified or tested. The following methods for architecture development are worth mentioning:

- Although meant for software development, the Rational Unified Process (RUP) (Jacobson et al., 1999) is of interest here, as it defines an iterative process, as opposed to the classical waterfall process, that realizes software by adding functionality to the architecture at each increment.
- The UN/CEFACT Modeling Methodology (UMM) is an incremental business process and information model construction methodology. The scope is intentionally restricted to business operations, omitting technology-specific aspects. The Business Collaboration Framework (BCF), which is currently under development,

will be a specialization of the UMM aimed at defining an enterprise's external information exchanges and their underlying business activities. See http://www.unbcf.org/index.html.

- The Chief Information Officers Council has created The Federal Enterprise Architecture Framework (FEAF) accompanied by a practical and useful manual for developing enterprise architecture for governmental organizations. See http://www.cio.gov. Other initiatives of the United States government include Technical Architecture Framework for Information Management (TAFIM) by the U.S. Department of Defense and the Treasury Architecture Development Process by the Department of the Treasure. See http://www.library.itsi.disa.mil/tafim.html and http://www.ustreas.gov/teaf/, respectively.
- The TOGAF Architecture Development Method (ADM), developed by the Open Group, provides a detailed and well-described phasing for developing an IT architecture. Version 8 of TOGAF entitled "Enterprise Edition" provides a framework and development method for developing enterprise architectures. See http://www.opengroup.org/architecture/togaf8.
- MEMO (Frank, 2002) is a method for enterprise modeling that offers a set of specialized visual modeling languages together with a process model, as well as techniques and heuristics to support problem specific analysis and design. The languages allow the modeling of various interrelated aspects of an enterprise.

Differences between these methods are partly due to historical reasons and partly due to differences in scope. First, they may or may not specify the detailed techniques, languages, or tools to be used in each phase. Second, they differ in the degree to which they encourage the repetition of various stages. Some methods take the "right first time" approach for each stage of development, while others promote iteration. Some methods favor the involvement of users during the whole process, while others limit user involvement to the early stages.

Modeling Support

The industry has produced a number of tools supporting architecture work within enterprises, especially in the area of modeling and modeling languages (for example, the ARIS toolset, the Rational tools, Metis, Enterprise Architect, System Architect, Testbed Studio, MEMO). Languages are an essential instrument for the description and communication of architectures, and languages and tools have evolved more or less hand in hand. In some cases, methodologies and frameworks have grown around and are supplied together with architecture support tools, for instance, in the Rational, ARIS (Scheer, 1994), Testbed (Eertink et al., 1999) and MEMO cases. In other cases, tool vendors have strived to endow their tools with new functionality in order to support frameworks (for example, System Architect was supplemented with a Framework Manager, which supports, among others, the Zachman and TOGAF frameworks) or other

modeling notations such as UML (Booch, Rumbaugh & Jacobson, 1999) or the IDEF family (IDEF, 1993), besides their own proprietary notations (for example, ARIS, System Architect). Languages and modeling notations are at the core of all these architecture support packages.

Most languages mentioned provide concepts to model specific domains, for example, business processes or software architectures, but they rarely model the high-level relationships between these different domains. We can illustrate this with Figure 1. In current practice, architectural descriptions are made for the different domains. Although, to a certain extent, modeling support within each of these domains is available, well-described concepts to describe the relationships *between* the domains are almost completely missing. Such concepts are essential to tackle the problems of business-IT alignment and architecture optimization in a systematic way.

Service Orientation in Enterprise Architecture

From the overview of enterprise architecture in the previous section, one can conclude that there are two main issues in enterprise architecture today:

1. The problem of alignment between the various architectures and
2. The lack of a guiding principle for overall optimization with respect to an enterprise's goals.

The service concept may provide an interesting direction to solve these issues. The idea of systems (applications or components) delivering services to other systems and their users is really starting to catch on in software engineering; witness, for example, this book. However, in other relevant disciplines, there is an increasing focus on services, too. In fact, economic development is to an increasing extent driven by services, not only in traditional service companies but also in manufacturing companies and among public service providers (Illeris, 1997). In the service economy, enterprises no longer convert raw materials into finished goods, but they deliver services to their customers by combining and adding value to bought-in services. As a consequence, management and marketing literature is increasingly focusing on service design, service management, and service innovation (for example, see Fitzsimmons & Fitzsimmons, 2000 or Goldstein et al., 2002).

Another upcoming area in which the service concept plays a central role is IT service management. This discipline is aimed at improving the quality of IT services and the synchronization of these services with the needs of their users (Van Bon, 2002).

Combining these three developments — the focus on services in management, the growing attention for service management, and the hype around Web services — convinced us that services should have a more prominent role in enterprise architecture.

In the rest of this section, we study what the impact of service orientation is on enterprise architecture and how it could potentially solve the two problems identified above.

The Service Concept

Given the central role of the service concept in this chapter, a clear and precise definition is required. This definition should make sense both in the business and in the IT domain. For a generic definition, we refer back to the seminal work by Vissers and Logrippo (1985) and Quartel et al. (1997), where a service is defined as "the observable behavior of a system (the service provider) in terms of the interactions that may occur at the interfaces between the system and the environment and the relationships between these interactions". The term *system* is used here in the widest sense, including both applications and organizational units.

The service concept is the result of a separation of the *external* and *internal* behavior of a system. As such, it should be self-contained and have a clear purpose from the perspective of its environment. The internal behavior, on the other hand, represents what is required to realize this service. For the *consumers* of a service, the internal behavior of a system or organization is usually irrelevant: they are only interested in the functionality and quality that will be provided.

Relevance and Benefits

One might ask why we should focus on services for architecting the enterprise and its IT support. What makes the service concept so appealing for enterprise architecture practice? First, there is the fact that the service concept is used and understood in the different domains making up an enterprise. In using the service concept, the business and IT people have a mutually understandable *language,* which facilitates their communication. Second, service orientation has a positive effect on a number of key differentiators in current and future competitive markets, that is, interoperability, flexibility, cost effectiveness, and innovation power.

Interoperability

Of course, Web services and the accompanying open XML-based standards are heralded for delivering true interoperability at the information technology level (Stevens, 2002). However, service orientation also promotes interoperability at higher semantic levels by minimizing the requirements for shared understanding: a service description and a protocol of collaboration and negotiation are the only requirements for shared understanding between a service provider and a service user. Therefore, services may be used by parties different from the ones originally perceived or used by invoking processes at various aggregation levels.

Flexibility

Interoperability and separation of internal and external behavior provide new dimensions of flexibility: flexibility to replace or substitute services in cases of failure, flexibility to upgrade or change services without affecting the enterprise's operations, flexibility to change suppliers of services, flexibility to reuse existing services for the provision of new products or services. This will create new opportunities for outsourcing, rendering more competition and more efficient value chains.

Cost Effectiveness

By focusing on services, many opportunities for reuse of functionality will arise, resulting in more efficient use of existing resources. In addition, outsourcing and competition between service providers will also result in a reduction of costs. From a more macroscopic point of view, costs will be reduced as a result of more efficient distribution of services in value chains.

Innovation Power

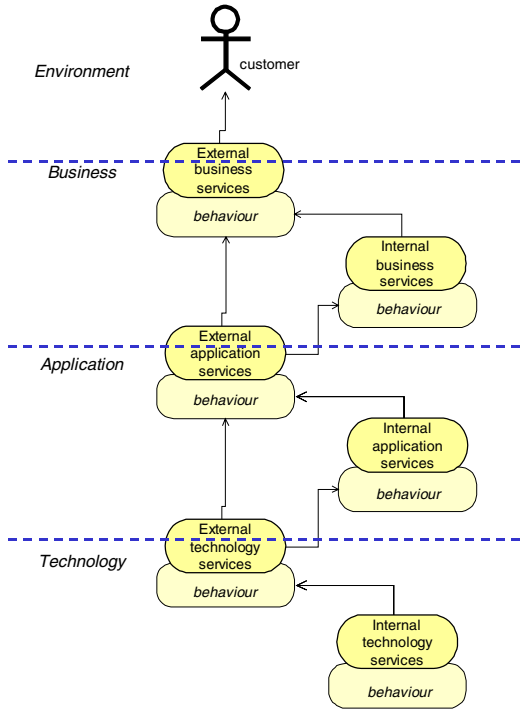
The ability to interoperate and collaborate with different partners, including partners not familiar to the enterprise, provides new opportunities for innovation. Existing services can be recombined, yielding new products and services, *ad hoc* liaisons with new partners become possible that exploit emerging business opportunities, and newly developed services can easily be advertised and offered all over the world, and integrated in the overall service architecture.

Finally, service orientation stimulates new ways of thinking. Traditionally, applications are considered to support a specific business process, which in turn realizes a specific business service. Service orientation also allows us to adopt a bottom-up strategy, where the business processes are just a mechanism of instantiating and commercially exploiting the lower-level services to the outside world. In this view, the most valuable assets are the capabilities to execute the lower-level services, and the business processes are merely a means of exploitation.

Introducing a Service Architecture

The wide applicability of the service concept to all levels of enterprise architectures paves the way for the introduction of a separate service architecture. The service architecture defines and relates all services of an enterprise. The enterprise is essentially regarded as a collection of interrelated services at various aggregation levels, or more precisely, a collection of abilities to instantiate services. These services can be business services or technical services, and the services can be both high-level aggregated services and low-level atomic services.

Figure 2. Service architecture: Hierarchy of services



A service architecture can be extracted from existing architectures by projecting out all (business, application, and technology infrastructure) services provided, or it can be developed separately. Services are typically grouped by type in a service architecture, as depicted in Figure 2.

The resulting hierarchy corresponds to the architectural layers (business, application, and technology) defined in the architectural framework in Figure 1. Each layer makes its external services available to the next higher layer. The external services of the higher layer may depend on services in the same or lower architectural layers. Business services, for example, may depend on external application services. Internal services are used within the same architectural layer; for instance, an application component may use services offered by another application component. Likewise, a business process may be viewed as comprising subprocesses that offer their services to each other and to the containing process. External business services could also be called *customer services,* that is, services offered to the (external) customers of the enterprise.

Naturally, within these different layers, services will have to be augmented with aspects specific to these layers. This might entail, for example, adding (representations of) service level agreements to services in the business layer or WSDL specifications to

describe the details of internal application services. However, central is the generic concept of a service as a business-relevant unit of behavior as it is exposed to the environment.

Guidelines

The development of a service-oriented enterprise architecture (SOEA) should be guided by the following principles in order to achieve the benefits listed previously.

Separation of Internal and External Behavior

Following the definition of the service concept, a service should only define the externally observable behavior of a system, not how that behavior is realized. Such encapsulation has long been a guiding principle in software development (for example, see Dijkstra, 1968). It provides a mechanism for being truly platform-independent for substituting different implementations with the same external behavior or interchanges different suppliers of services.

Definition of the Meaning of Services in External Context

The definition of the external behavior should be in terms of the invoking processes, systems, or users, making the added value and possible uses of the service explicit for service consumers. It encourages cross-domain thinking and design, reducing the semantic gaps between domains. It also facilitates the communication between stakeholders from these different domains, such as business analysts and software architects.

Minimization of Shared Understanding

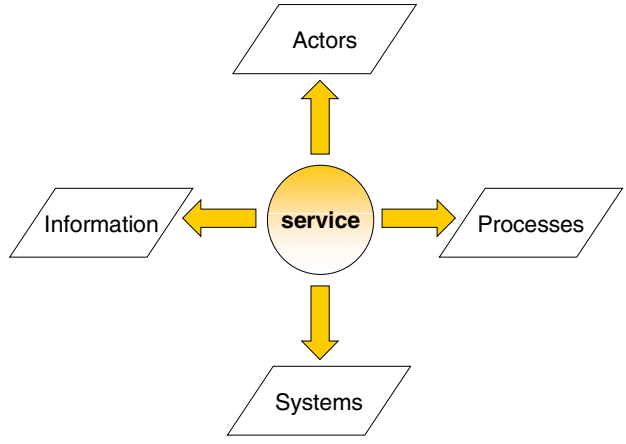
Services should be specified such that (potential) users require a minimum amount of information to understand the external behavior of the service and with a minimum number of handles to operate the service for these purposes. Minimizing the shared understanding is an enabler for the second guiding principle, as it reduces gaps from different perspectives.

These guiding principles provide directions for the development of services and interoperability in all sorts of domains. They hold for the development of infrastructural services, application services, business processes, and business functions.

Architectural Alignment

Once a service architecture is established, it can be used as a vehicle for achieving architectural alignment or as the starting point for optimization or redesign of the

Figure 3. Services as a pivotal construct linking different architectural domains



enterprise. As we explained in the introduction, an enterprise architecture is not a single entity but a collection of all relevant architectures making up the enterprise. One of the main challenges in enterprise architecture is to relate and align all these architectures with each other and with the overall goals and objectives of the enterprise.

Having a service architecture makes it easier to relate the various architectures to one another. We have already shown that a service architecture provides insight in the dependency relationships between an enterprise's services at different levels. The service architecture thus provides a global overview of the functioning of an enterprise, which can be used to play *what-if* games. For example, what is the impact if a certain application service is removed, outsourced, or upgraded?

In addition, services can serve as linking pins between the concepts used in other architectural domains. A service is used by and provided by *actors* fulfilling *roles* in the organization domain. A service is invoked by and implemented by *business processes* in the process domain or *application components* in the application domain. And, a service requires and processes *information* and *data* from the information and data domains, respectively.

It may seem too much of a simplification to use only one construct for the complicated matter of enterprise design, but service-oriented architecture does not mean that there is no longer a place for classical distinctions between roles, processes, applications, information, and so forth. Rather, in service-oriented architectures there is one pivotal construct (the service concept) between different aspects and between different levels of aggregation. The different types of architecture arise as aspect views for the service-oriented architecture. Thus, SOEA imposes a direct correlation between business processes and application services, improving governance and maintainability, while simplifying the development of new services from existing ones.

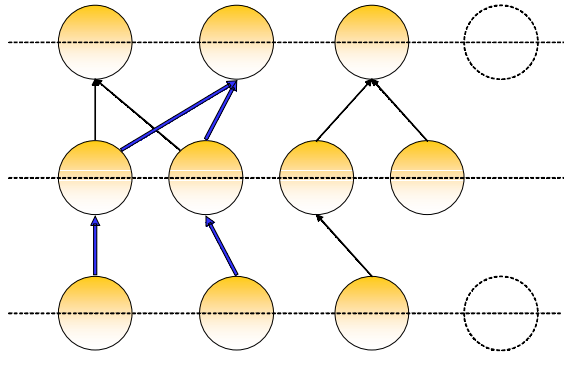
Service Optimization

Currently, optimizing an enterprise architecture, that is, improving performance, quality, or cost effectiveness, is typically done locally within one type of architecture. For example, an enterprise may decide to change its business processes to improve its efficiency. This change might require changes in the supporting applications and IT infrastructure and lead to changes in the organization. In this way, the enterprise can, at most, achieve a locally optimal situation because the resulting changes could have a detrimental impact on the efficiency of the applications and on the effectiveness of the organization as a whole. Our hypothesis is that services could provide the overall optimizing concept currently lacking in enterprise architecture. This hypothesis still requires validation but is derived from the reasoning that most enterprises today belong to the service industry and compete on service levels. Therefore, their goal generally is to provide the best possible quality of service.

There are two ways in which services could guide optimization. The first way is to start from a particular service and analyze how this service is supported in different architecture domains. This is illustrated in Figure 3. The challenge is to design processes, organizational structures, and information systems to effectively support the selected service.

Even more challenging is the optimization of the entire set of services across architectural domains and aggregation levels. Consider the archetypical corporate service architecture as illustrated in Figure 4. An end-user service (top level) depends on lower-level services for its service delivery, which again may depend on service at an even lower level. On the other hand, lower-level services can be used in different end-user services. In general, to attain the goal of cost reduction, lower-level services should be more generic than higher-level services, limited in number, and used by as many higher-level services as possible. However, the more a lower-level service is used by higher-level

Figure 4. Corporate service architecture with different layers of services (circles) and dependencies (arrows) (The thick arrows represent a particular set of services that are required for one of the end-user services.)



services, the more difficult it will be to change this service. To retain flexibility, decomposition of such services may be preferable.

From this example, it is clear that it is a real challenge to define guidelines for optimal service architectures that specify decomposition and reuse to achieve the combined goals of increased flexibility, costs effectiveness, and innovation power.

One could argue that this problem is not different from the problems of component-based design. In fact, it is similar, but it is now applied to the entire service architecture of the enterprise. In this field, it provides a new concept for enterprise optimization. Current optimization strategies start from classical domains, for example, business processes, products, customers groups, as starting points to perform an optimization strategy. In service orientation, the services are the primary building blocks of organizations. Because the service concept intrinsically refers to interoperability, flexibility, and reuse, as argued in the Benefits and Relevance section, the promise of service orientation is that optimization strategies based on services will lead to more optimal business architectures for the envisioned competitive business world.

Modeling Support for Service-Oriented Enterprise Architecture

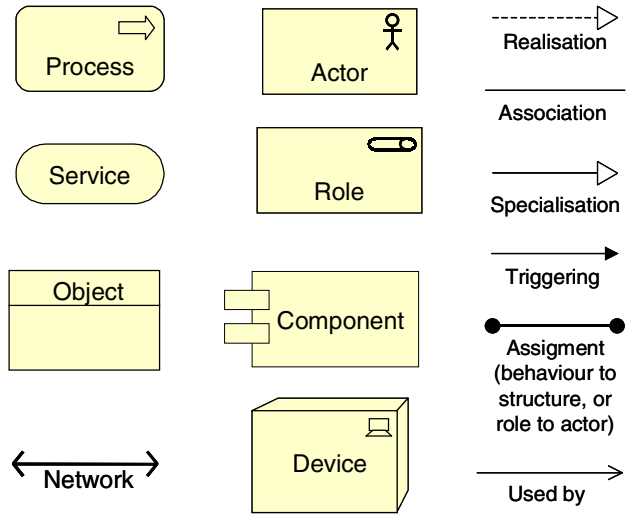
In order to facilitate a service-oriented approach to enterprise architecture, a high-level modeling language is needed in which the different conceptual domains can be described at a sufficiently abstract level. Such a language, in which the service concept plays a central role, is being developed in the ArchiMate project (Jonkers et al., 2003). The objective of the ArchiMate language is to define relationships between concepts in different architectures, the detailed modeling of which may be done using other standard or proprietary modeling languages.

Concepts in the ArchiMate language currently cover the business, application, and technology layers of an enterprise. For each layer, concepts and relations for modeling the information, behavior, and structure aspects are defined. Services offered by one layer to another play an important role in relating the behavior aspects of the layers. The structural aspects of the layers are linked through the interface concept and the information aspects through realization relations.

Figure 5 illustrates the main concepts in the ArchiMate architectural modeling language. The concepts of this language hold the middle between the detailed concepts that are used for modeling individual domains, for example, the UML for modeling software. For a more complete definition of the language, we refer to the ArchiMate project archive at http://archimate.telin.nl/.

In order to illustrate our approach to service-oriented enterprise architecture and to using services for architectural alignment, we have developed an example enterprise architecture for an imaginary insurance company (Figure 6). It illustrates the use of services to relate the infrastructure layer, the application layer, the business process layer, and the environment. The insurant and insurer roles represent the client and insurance company (ArchiSurance), respectively. Invocation of the claims registration service by the

Figure 5. Concepts of the ArchiMate architectural modeling language

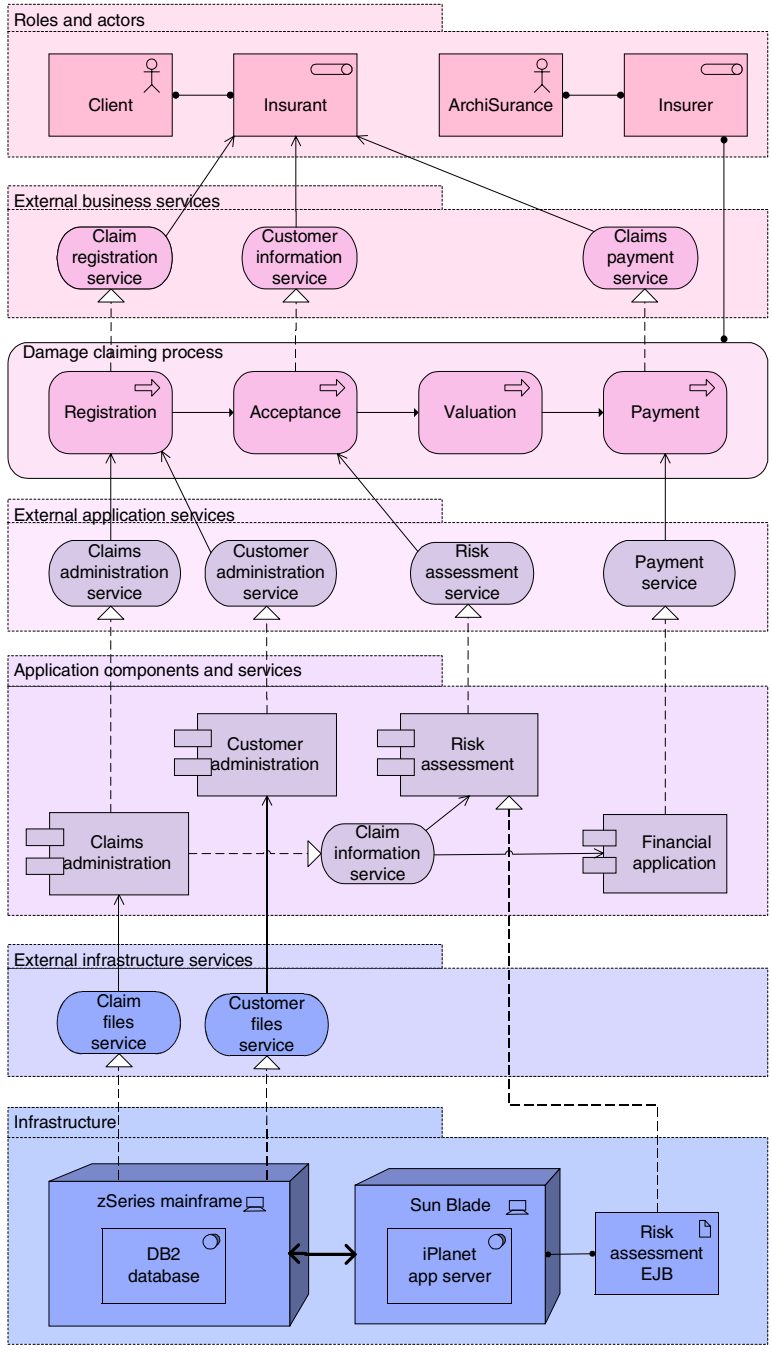


insurant starts the damage claiming process. The insurant is informed whether the claim is accepted, and, if so, the insurant receives a payment. Interaction between business processes and organizational roles is through business services. Thus, services connect the process architecture and the organization architecture. Likewise, application services relate the business process architecture to the application architecture. The automated part of each business process is provided by an external application service. These application services are realized by application components. Finally, the technology layer consists of a number of infrastructure elements, such as a mainframe and an application server, which execute application components and provide services to the application layer.

In the example, a high-level overview of an entire enterprise is shown in a single integrated and well-defined model. Admittedly, our example is simple; in reality, such a model would be much larger, requiring techniques for selecting and visualizing the elements that are relevant for a particular stakeholder.

An important advantage of the service concept is that it can be interpreted by both business and IT people. This model can be used by, for example, both a manager requiring the *big picture* and a software engineer that implements an application component and needs to know the context of this component. Thus, by using such a service-centric model, different stakeholders can better understand each other. Within each specific domain, this high-level model serves as a starting point for more detailed descriptions.

Figure 6. Example of a service-oriented enterprise architecture (For more details on the notation used in this example, see Van Buuren et al., 2003.)



Future Trends

Currently, most developments with respect to service orientation are taking place on the technology front. Web services standards continue to emerge at a dazzling speed. Unfortunately, there is little ongoing effort to revise methods for enterprise architecture in the light of service orientation.

Emerging Technologies

Web Services

Web services are a relatively young technology in full development, sustained by a rapidly evolving set of industry standards. Their broad acceptance is guaranteed by the global status of organizations such as W3C, UN-CEFACT, UDDI.org, OMG, and OASIS that lead the standardization work in this field. However, there are problems (for example, security, interoperability, availability, and reliability) that are not yet completely addressed, and therefore, most of these standards should be seen as work in progress.

We are witnessing a strong competition for the leading positions in the Web services market. The list of competing companies include big supporters of Web services such as Microsoft with its .Net strategy, IBM with its WebSphere, its "business on demand" framework and its patterns for e-business, Novell with its DENIM (Directory-Enabled Net Infrastructure Model) cross-platform infrastructure, Sun with its ONE (Open Net Environment), BEA Systems with its WebLogic, and many others.

Grid Services

A parallel development in service orientation is the ability to access ICT resources, such as computing power, storage capacity, and ICT devices as services over the Internet. With *devices,* we refer to everything that can be shared via the network. It includes scientific devices such as radio telescopes or MRI scanners, as well as your home video camera or your PDA. This development has its origin in e-science environments (computing grids), but also has large potential for a variety of other application areas like healthcare, education, finance, life sciences, industry, and entertainment. The idea is that in the near future, a user or a company can simply plug into the wall to get access to commoditized computing and storage services. In analogy with electricity provisioning over the *Power Grid,* this next generation service infrastructure is called the Service Grid. This will give large and small organizations access to ICT resources currently out of reach.

For grid development, the Global Grid Forum (http://www.ggf.org) leads standardization. The integration with business requirements is addressed in the OGSA (open grid services architecture) and OGSI (open grid services infrastructure) working groups. These primarily concern basic integration of grid computing concepts with Web service technology.

The grid service developments strengthen the impact of service orientation on business architectures because they extend the application of Web service technology to the domain of utility computing and ASP, while its focus on sharing of ICT resources will have additional impact on the way ICT infrastructure services are managed within organizations.

Real-Time Business Service Management

Tool vendors like BMC (http://www.bmc.com) recognize the importance of integrating real-time IT service management with operational business processes and customer services. They provide tools that propagate events at the IT level to process owners and customers; the other way around, problem reports from users and customers can be propagated to the IT service level. Such integrations should offer operational business-IT alignment giving insight into real-time performance and service levels. These developments create a strong case for service-oriented methods since they apply service orientation in real-time operational service management allowing services to be used for online decision making and problem solving.

Adoption Patterns

Drivers for Adoption

Although many enterprises are very reluctant in embracing what might appear to be yet another IT hype, there are some that have already started to implement Web services and gained some practical experience in understanding how and where they can use them. In a survey concerning the usage of Web services by early adopters of SOA, Wilkes (2003) has identified several incentives to move toward SOA that are worthwhile to be mentioned here because they express a position that is becoming a trend: SOA is a strategic decision, it delivers more flexible solutions for business, is more practical and more cost effective than the existent architectures, is compliant with existing and emerging standards, delivers a more practical solution for IT, allows for easy business process reengineering and optimization, and is device- and platform-independent.

Maturity Model

Since both service-oriented development methods and technology and standards surrounding Web services are still very much under development, we may ask ourselves what the future of service-oriented architectures will be like and when all these developments will finally lead to a mature, widely accepted and stable approach for organizations to proceed with such environments.

One outlook that tries to give a realistic prognosis for the near future is that of the Stencil Group (Sleeper & Robins, 2002). Their forecast with respect to the growth of the Web

services market emphasizes three phases: the first will cover the "organic adoption of Web services tools and standards" (2001-2003), the second, the "systematic deployment of services infrastructure" (2002-2006), and the third, the "pervasive use of services in collaborative business processes" (2005 and beyond).

The vision of Sleeper and Robins is partly confirmed by Sprott's (2003) maturity model that identifies four phases on the way of SOA toward maturity: *early learning* (experimental, mostly internal, focused on better application integration) which is happening now, *integration* (still internal, business process oriented and based on a more mature understanding of SOA) that will start probably in 2004 and will take three years, *reengineering* (services used across organizations and implemented as part of business products, both internal and external) that will start in 2005, and finally, *maturity* (ubiquitous and federated services, service consumer ecosystems). Sprott does not predict when we should expect maturity.

Road Maps for Moving to a Service-Oriented Architecture

There are numerous resources available that discuss what Web services are, how to implement or use them, what their benefits are in terms of costs, ROI, flexibility, and architecture integration. However, in contrast with the literature addressing classical enterprise and software architectures, one can hardly find well-structured methodologies, frameworks, or best practices that might assist enterprise architects during the complex migration process from classic enterprise architectures to SOEA and Web services. This is not a total surprise because such instruments are usually developed as soon as the expertise gathered in the application of new ideas and technologies in real environments reaches a certain critical mass, which is not yet the case with SOEA: there are few examples of fully developed and mature service-oriented enterprise architectures. However, we can refer to a few *roadmapping* initiatives in the area of SOA.

CBDi Forum has proposed "the Web Services Roadmap", which is in fact a collection of articles (see http://roadmap.cbdiforum.com/) focused on practical guidance for organizations adopting SOA and Web services. This Roadmap is structured around several "streams," which provide a division of the migration process into its main activities:

- The "Plan & Manage" stream involves activities related to the development and coordination of common policies and practices between the parts of the new federated environment, enabled by SOA.
- The "Infrastructure" stream offers guidance on the strategies, activities, and timing involved in the transition of the existing infrastructures.
- The "Architecture" stream covers the integration of Web services into core business processes, as the units of reuse across an organization.
- The "Process" stream deals with the "service life cycle" seen as a collaborative process between the Supplier and the Consumer of services.
- The "Projects" stream briefly refers to five service project profiles.

The second approach belongs to The Stencil Group (Robins et al., 2003). The first part of this report is dedicated to the business objectives and technical patterns emerging from the experiences of early adopters. In the second part, the authors propose a "Web service scorecard", intended to become a decision-making tool for new adopters. The scorecard is based on the findings of a study of 50 organizations and is divided in two parts: the first half should be used for the assessment of the enterprise IT strategy with respect to Web services and the second half for the examination of the services that are fit for a specific project. Apart from the scorecard, the paper discusses a four-step plan of "how to get started".

The third contribution originates from ZapThink, in the form of "ten emerging best practices" (Bloomberg, 2003) and a "path for SOA implementation" (Schmelzer & Bloomberg, 2003). These best practices give general common sense indications (such as "Encapsulate existing/legacy functionality" or "Compose atomic Services into coarse-grained business Services") that might be useful principles to follow during the design of service architectures. No time line or methodological steps are suggested. Nevertheless, the authors strongly encourage the use of principles and techniques of agile methodologies. In contrast with the best practices, "the path for SOA implementation" is an attempt to express in a graphical manner the most important moments toward the maturity of a SOEA. The process is divided into four phases: Point-to-point integration, Internal SOAs, B2B process-driven services, and the on-demand enterprise.

Conclusion

Service orientation is a new paradigm, not only for software engineering but also for the broader topic of enterprise architecture. Service-oriented enterprise architecture (SOEA) introduces the idea of a service architecture, which facilitates alignment between the various architectural domains. In addition, services and the service architecture are useful starting points for synchronizing an enterprise's design with its goals.

Future research will have to determine whether service orientation really can deliver on all its promises of increased interoperability, flexibility, and innovation power. Some organizations are already starting to experiment with service-oriented enterprise architectures. We will carefully monitor their experiences to verify if they can indeed improve their competitive power.

Thus far, SOEA is merely an appealing idea. It will have to be operationalized by concrete methods and techniques that are centered around the service concept. Work to this end is, for example, taking place within the ArchiMate project. We expect this work to generate many more issues and questions related to implementing a service-oriented architecture. For example, what aspects of a service should be specified? How should a service be specified? How can service management profit from an explicit service architecture? And so forth.

References

Arkin, A. (2002). Business Process Modeling Language, BPMI.org. Retrieved August 10, 2004, from *http://www.bpmi.org/bpmi downloads/BPML1.0.zip*

Biemans, F. P. M., Lankhorst, M. M., Teeuw, W. B. & van de Wetering, R. G. (2001). Dealing with the complexity of business systems architecting. *Systems Engineering, 4*(2).

Bloomberg, J. (2003). Ten emerging best practices for building SOAs, ZapThink. Retrieved August 10, 2004, from *http://searchWebservices.techtarget.com/originalContent/1,289142,sid26_gci882714,00.html*

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language user guide.* Addison-Wesley.

Buchanan, R. D., & Soley, R. M. (2002). Aligning enterprise architecture and IT investments with corporate goals, OMG.

Dijkstra, E. W. (1968). Structure of the 'THE'-Multiprogramming system. *Communications of the ACM, 11*(5), 341-346.

Eertink, H., Janssen, W., Oude Luttighuis, P., Teeuw, W., & Vissers, C. A. (1999, September). *A business process design language.* Proceedings of the 1st World Congress on Formal Methods, Toulouse, France.

Eriksson, H.-E., & Penker, M. (2000). *Business modeling with UML: Business patterns at work*. New York: Wiley.

Ferris, C., & Farrell, J. (2003). What are Web services? *Communications of the ACM, 46*(6).

Fitzsimmons, J. A., & Fitzsimmons, M. J. (2001). *Service Management* (3rd ed.). New York: McGraw-Hill.

Fitzsimmons, J. A., & Fitzsimmons, M. J. (2000). *New service development: Creating memorable experiences.* Thousand Oaks, CA: Sage.

Frank, U. (2002). *Multi-perspective Enterprise Modeling (MEMO) - Conceptual framework and modeling languages.* Proceedings of the Hawaii International Conference on System Sciences (HICSS-35), Honolulu.

Garschhammer, M., Hauck, R., Kempter, B., Radisic, I., Roelle, H., & Schmidt, H. (2001). The MNM service model - Refined views on generic service management. *Journal of Communications and Networks, 3*(4), 297-306.

Goldstein, S. M., Johnston, R., Duffy, J., & Rao, J. (2002, April). The service concept: The missing link in service design research. 121-134.

IDEF. (1993). *Integration Definition for Function Modeling (IDEF0) Draft* (Federal Information Processing Standards Publication No. FIPSPUB 183.) U.S. Department of Commerce, Springfield, VA.

IEEE. (2000). IEEE Computer Society, *IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, Oct. 9, 2000.

Illeris, S. (1997). *The service economy: A geographical approach.* Wiley.

ITU. (1996). ITU Recommendation X.901 | ISO/IEC 10746-1: 1996, Open Distributed Processing - Reference Model - Part 1: Overview.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development method.* Addison-Wesley.

Jonkers, H., et al. (2003). *Towards a language for coherent enterprise architecture descriptions.* Proceedings of the 7th International IEEE Enterprise Distributed Object Computing Conference (EDOC).

Kramer, J., & Finkelstein, A. (1991). A configurable framework for method and tool integration. *Software Development Environments and CASE Technology*, 233-257.

Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering, 26*(1), 70-93.

Menor, L. J., Tatikonda, M. V., & Sampson, S. E. (2002). New service development: Areas for exploitation and exploration. *Journal of Operations Management, 20*(2), 135-157.

Quartel, D. A. C., Ferreira Pires, L., van Sinderen, M. J., Franken, H. M., & Vissers, C. A. (1997). On the role of basic design concepts in behavior structuring. *Computer Networks and ISDN Systems, 29*(4), 413-436.

Robins, B., Sleeper, B., & McTiernan, C. (2003). Web services rules: Real-world lessons from early adopters business-technology solutions, The Stencil Group. Retrieved August 10, 2004, from *http://www.stencilgroup.com/ideas/reports/2003/wsrules/*

Scheer, A.-W. (1994). *Business process engineering: Reference models for industrial enterprises* (2nd ed.). Berlin: Springer-Verlag.

Schmelzer, R., & Bloomberg, J. (2003). ZapThink's path to service-oriented architecture implementation poster. Retrieved August 10, 2004, from *http://www.zapthink.com/report.html?id=ZTS-GI102*

Sleeper, B., & Robins, B. (2002). The laws of evolution: A pragmatic analysis of the emerging Web services market, The Stencil Group. Retrieved August 10, 2004, from *http://www.stencilgroup.com/ideas_scope_200204evolution.pdf*

Sowa, J. F., & Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM Systems Journal, 31*(3), 590-616.

Sprott, D. (2003). A Web services maturity model: A strategic perspective for technology and business planning. Retrieved August 10, 2004, from *http://roadmap.cbdiforum.com/reports/maturity/index.php*

Stephenson, J. (2003). Roadmap report - UN/CEFACT move into enterprise architecture space. Retrieved August 10, 2004, from *http://www.cbdiforum.com/secure/interact/2003-10/un-cefact.php3*

Stevens, M. (2002). Service-oriented architecture introduction, part 1. Retrieved August 10, 2004, from *http://www.developer.com/design/article.php/1010451*

Van Bon, J. (Ed.). (2002). *IT service management: An introduction.* ITSMF.

Van Buuren, R. (Ed.). (2003). In S. Hoppenbrouwers, H. Jonkers, & M. M. Lankhorst, *Architecture language reference manual,* TI/RS/2003/030, (ArchiMate/D2.2.2b). Enschede: Telematica Instituut. Retrieved August 10, 2004, from *https://doc.telin.nl/dscgi/ds.py/Get/File-31626/*

Vissers, C. A., & Logrippo, L. (1985). The importance of the service concept in the design of data communications protocols. In M. Diaz (Ed.), *Protocol specification, testing and verification V* (pp. 3-17). North-Holland.

Wegmann, A. (2003). *On the systemic enterprise architecture methodology (SEAM).* Proceedings of the International Conference on Enterprise Information Systems (ICEIS 2003), Angers, France.

Wilkes, L. (2003). Web services usage survey. Retrieved August 10, 2004, from *http://www.cbdiforum.com/bronze/Webserv_usage/Webserv_usage.php3*

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal, 26*(3), 276-292.

Chapter VIII

A Method for Formulating and Architecting Component- and Service-Oriented Systems

Gerald Kotonya
Lancaster University, UK

John Hutchinson
Lancaster University, UK

Benoit Bloin
Lancaster University, UK

Abstract

This chapter describes a negotiation-driven method that can be used to formulate and design component and service-oriented systems. Component and service-oriented development are increasingly being promoted in literature as rapid low-cost strategies for implementing adaptable and extensible software systems. In reality, these development strategies carry significant risk throughout the system life cycle. The risks are related to the difficulty in mapping requirements to component and service-based architectures, the black-box software used to compose the systems, and the difficulty in managing and evolving the resulting systems. These problems underscore the need for software engineering methods that can balance aspects of requirements with

business concerns and the architectural assumptions and capabilities embodied in software components and services.

Introduction

The development of component and service-oriented systems share several characteristics (Bennett & Gold, 2003; Szyperski, 2002). Both approaches are based on *development with reuse* and are therefore constrained by the availability of suitable off-the-shelf software components and services. In both cases, negotiation is central to achieving a balanced solution. In both cases, the design of the interface is done such that the software component or service exposes a key part of its definition. In general functional terms, there is little difference to the consumer between reusing an existing internal component or service and buying or renting an external component or services. Differences arise in the nature of the applications and how they are composed (Szyperski, 2001).

Component-based development proceeds by composing software systems from prefabricated components (often third-party black-box software) (Brown & Wallnau, 1998; Szyperski, 2002). A typical component-based system architecture comprises a set of components that have been purposefully designed and structured to ensure that they fit together (that is, have *pluggable* interfaces) and have an acceptable match with a defined system context. Service-oriented development proceeds by integrating disparate heterogeneous software services from a range of providers (Cerami, 2002; Layzell et al., 2000; Stal, 2002). A service-oriented architecture is a means of designing software systems to provide services to either end-user applications or other services through published and discoverable interfaces. A typical service-oriented architecture comprises a service requestor, service provider, and service broker (registry) that interact through standard messaging protocols (for example, HTTP and SOAP) that support the publishing, discovery, and binding of services. However, the diverse nature of software systems means that it is unlikely that systems will be developed using a purely service or component-based approach (Kim, 2002; Kotonya & Rashid, 2001). Rather, a hybrid model of software development where components and services coexist in the same system is likely to emerge.

This chapter describes a method for software system development, COMPOSE (COMPonent-Oriented Software Engineering), that extends the notion of *service* to requirements definition to provide a framework for mapping requirements to hybrid component/service-oriented architectures. The method incorporates negotiation as a key process activity to balance aspects of system requirements and business concerns with the architectural assumptions and capabilities embodied in software components and services. The focus of the method is on system formulation and design. However, the method also provides *hooks* that allow it to be extended to system composition and management.

Background

Component and service-oriented development poses many challenges to organizations intending to adopt them:

- Traditional software development approaches are unsuitable for developing component and service-oriented systems (Boehm & Abts, 1999):
  - In the waterfall model, requirements are identified at an earlier stage and the components chosen at a later stage. This increases the likelihood of the components not offering or supporting required features.
  - Evolutionary development assumes that additional features can be added if required. However, the inaccessibility of component code prevents developers from adjusting them to their needs.
- There is a general lack of analysis tools that support *development with reuse* (particularly black-box development).
  - Limited specification provided with software components and services makes it difficult to predict how the systems that were built using them behave under different loads and application contexts.
  - The features supported by the components and services may vary greatly in quality and complexity. This complexity together with the variability of application contexts means that specifications delivered with black-box software are likely to be incomplete or inadequate.
  - The design assumptions of black-box software are largely unknown to the application builder.
- There is a general lack of methods for mapping functionality to services and for grouping services into logical domains (Nadhan, 2003).
  - Proper identification of services and determination of corresponding service providers is a critical first step in architecting a service-oriented solution. It is worth noting that in today's world, similar business functions could very well be provided by multiple systems within (and external to) the enterprise. The architectural framework adopted must provide a means for service rationalization. This involves careful analysis of all the systems and applications providing the given business function to ensure a more consistent delivery of services.
  - For most organizations, a key business objective is that a given service operates in an ideal location for the service. However, distributed architectural solutions can result in critical, often sensitive business data, being spread across multiple applications and service providers. It is important that the partitioning process takes these factors into account.

  - Service grouping or clustering has a direct influence on many important system characterics such as load balancing, access control, performance, maintainability, safety management, proxy simulation, vertical or horizontal partitioning of business logic. However, it is often a serious challenge for business units and technology centers within an enterprise to come to a consensus on an appropriate definition of service domains.
- There is a need for effective mechanisms to support service orchestration. A given service exists because there is at least one instance of a service consumer initiating the request for that service. In some scenarios, however, a service may have to invoke many other services to fulfil service consumers' original request. However, complex scenarios can involve recursive invocation of multiple services and, in some extreme cases, interdependent invocation of multiple services, which could result in a deadlock.
- Regardless of the way service domains are defined within an enterprise, need is likely to emerge for creating new services and modifying existing ones. Current service models do not define schemes for monitoring, defining, and authorizing the changes to existing suites of services supported within the enterprise.

These problems underpin the need for software engineering processes and methods that:

- Can balance aspects of system requirements, business and project concerns, with the assumptions and capabilities embodied in off-the-shelf software components. Current methods for *development with reuse* have focused on specific development activities (for example, component selection and component specification) rather than the process. This has obscured the correspondence between the different activities and made it difficult to achieve a balanced solution.
- Can support hybrid component/service-oriented development to leverage their different design strengths. There is a general lack of software engineering approaches that support this kind of hybrid development.

COMPOSE is a service-based, negotiation-driven method that supports a hybrid development approach. Unlike traditional methods, COMPOSE is not a closed approach but a framework for integrating different methods and techniques. These are mapped onto a generic *development with reuse* process that supports development, verification, and negotiation (Kotonya, Sommerville & Hall, 2003).

Development Process

COMPOSE is mainly intended to support black-box development but makes allowances for white-box development where black-box development is not feasible. Figure 1 shows

the four-phase development process. The planning phase sets out justification, objectives, strategies (methods and resources to achieve development objectives), and tactics (start and end dates and tasks with duration) for the development project. The development phase implements the agenda set out in the planning phase. The first step in application development is requirements engineering. This often starts with requirements elicitation, followed by requirements ranking and modeling (as system services). The requirements process is constrained by the availability of potentially suitable components and services as well the nature of the application.

The design stage partitions the service descriptions into abstract subsystem blocks with well-defined interfaces. Subsystems are replaced with concrete software components and services at the composition stage. Beyond this stage, the system goes into a management cycle. Like the requirements stage, the design stage proceeds in tandem with the verification and planning phases and may iterate to the requirements stage from time to time.

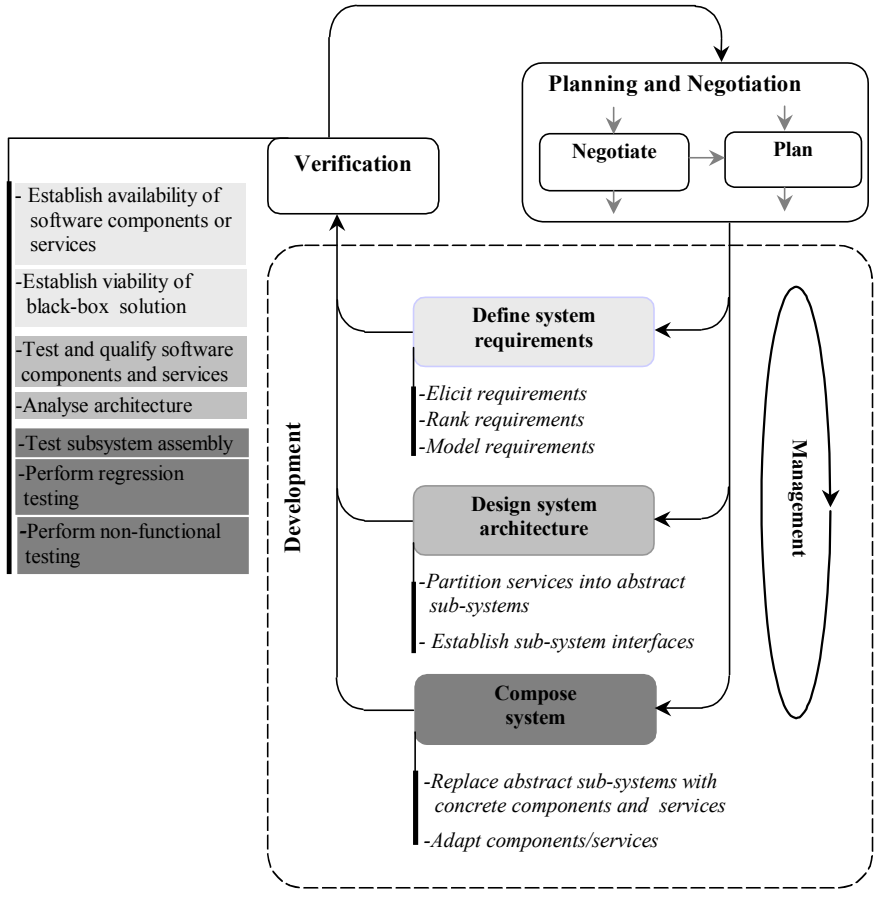
The verification phase is intended to ensure that there is an acceptable match between selected software components and services and the system being built. This is important because the perception of software quality may vary amongst software component producers and service providers. A matching color scheme has been used to show the correspondence between the different development stages and aspects of verification that apply to them. At the requirements stage, verification is used to establish the availability of suitable software components and services and the viability of a reuse-driven solution. At the design stage, verification is concerned with ensuring that the design matches the system context (system characteristics, such as requirements, cost, schedule, operating, and support environments). This may require detailed black-box testing of the software components and architectural analysis. At the composition stage, verification translates to design validation through subsystem, assembly, and system testing. The negotiation phase provides a framework for reviewing aspects of system development and for trading-off competing attributes.

The Method

Requirements Engineering

The principal challenge in defining system requirements for *development with reuse* is to develop requirements models and methods that allow us to balance aspects of requirements with the assumptions and capabilities embodied in software components and services. However, few approaches address themselves to this challenge. Vigder, Gentleman, and Dean (1996) propose that system requirements should be defined according to what is available in the marketplace and that organizations should be flexible enough to accept off-the-shelf solutions when they are proposed. They note that overly specific requirements preclude the use of off-the-shelf solutions and should be avoided. This is a reasonable assumption; however, most systems have requirements that are unavoidably *specific,* for example, critical systems.

Figure 1. System development process



Ncube and Maiden (1999) propose an approach in which the requirements process is tightly integrated with a process for selecting off-the-shelf products. Central to the PORE (Procurement-Oriented Requirements Engineering) approach is an iterative process in which candidate products are tested for fitness against increasing levels of requirements detail. PORE's singular focus on component selection has been criticized for ignoring system level concerns and the important role architecture plays in formulating the requirements for these kinds of system (Kotonya et al., 2002).

Our proposed solution interleaves requirements definition with negotiation and software component verification. Negotiation ensures that there is an acceptable trade-off between the capabilities embodied in components or service, aspects of requirements, and critical architectural concerns. Verification serves three objectives. In the early stages of the requirements definition, it is useful as a coarse filter for establishing the

availability of suitable software components or services. In the later stages of requirements, verification may be used to establish how well selected software components or services match the desired system functionality and constraints. Verification may also be used at the requirements stage to provide project managers with an indication of the viability of a black-box solution.

The requirements approach used in COMPOSE has three iterative steps interleaved with component verification, negotiation, and planning:

1. Requirements elicitation
2. Requirements ranking
3. Requirements modeling

Eliciting Requirements

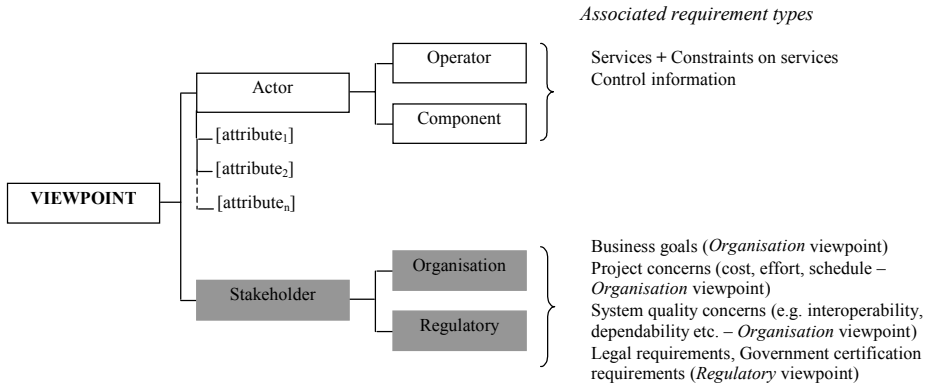
All requirements methods must address the basic difficulty of identifying the problem domain entities for the system being specified. The majority of methods provide little guidance in this, relying instead on the method user's judgment and experience. Our approach is based on the notion of *viewpoints*. Viewpoints correspond to requirements sources which comprise end-users, systems interfacing with the proposed system, organization, and external concerns (Kotonya, 1999). Our approach provides some help to the system developer in the critical step of viewpoint identification. We have generalized potential requirements sources into a set of viewpoints classes that can be used as a starting point for finding viewpoints specific to the problem domain.

Figure 2 shows the abstract viewpoint tree used as a starting point for eliciting system requirements. The root of the tree represents the general notion of a viewpoint. Information can be inherited by subclasses, so global requirements are represented in the more abstract classes and inherited by subclasses.

We have identified the following abstract viewpoints:

- *Actor viewpoints* are analogous to clients in a client-server system. The proposed system (or required component) delivers services (functional requirements) to viewpoints, which may impose specific constraints (nonfunctional requirements) on them. Actor viewpoints also pass control information and associated parameters (represented by viewpoint *attributes*) to the system. There are two main types of *Actor* viewpoints:
  - *Operator viewpoints* map onto classes of users who interact with the proposed system. They represent frequent and occasional users of the system.
  - *Component viewpoints* correspond to software components (or subsystems) and hardware devices that interface with the proposed system.
- *Stakeholder viewpoints* vary radically from organizational viewpoints to external certification bodies. Stakeholders are entities that do not interact directly with the intended system but which may express an interest in the system requirements.

Figure 2. Abstract viewpoint structure



These viewpoints often generate requirements that affect the way the system is developed. Stakeholder viewpoints provide a mechanism for expressing critical *holistic* requirements, which apply to the system as a whole although they may also generate requirements that affect a subset of its services or functionality. Stakeholder generated requirements reflect essential system characteristics such as dependability (that is, safety, reliability, security, and performance) but may also correspond to critical business and project objectives, such as resources, schedule, cost, and standardization.

A viewpoint template has the following structure:

Viewpoint id: <A unique viewpoint identifier>

Type: <Viewpoint type (for example, operator, system, component, organization, regulatory, and so forth)>

Attribute: <An optional set of data attributes for the *Actor* viewpoint>

Role: <Role of the viewpoint in the system>

Requirements: <Set of requirements generated by the viewpoint>

History: <Development history>

A requirement can be considered at different levels of abstraction to allow for scoping and ease of understanding. A requirement template has the following structure:

Requirement id: <Requirement identifier>

Rationale: <Justification for requirement>

Description: <Natural language definition>|<Service description>|<Other description>

Levels of abstraction may map the requirement description to different representations and levels of detail.

Ranking Requirements

Ranking techniques range from simple weighted schemes that are concerned with a single requirement aspect to multiattribute schemes that take into account several requirement aspects such as benefit, effort, and risk (Karlsson & Ryan, 1997; Lootsma, 1999; Saaty, 1980). However, most of these schemes are intended for custom development and are unsuitable for component-based development. They often require that detailed requirements to be formulated early, which is inappropriate for component-based development. It would also be difficult to provide useful estimates for system aspects, such as risk and effort at this early stage without prior knowledge of suitable components and services. COMPOSE uses requirement *benefit* as a basis for ranking requirements and solution-dependent aspects, such as effort and risk that are deferred to the component verification stage. Requirement benefit can be categorized as essential, important, and useful. The output from the ranking process is a list of prioritized requirements that together with potential components and services form input to the component verification process.

Modeling Requirements

The concept of a service is common to many areas of computing, including digital libraries, distributed computing, data management, and electronic commerce (Arsanjani et al., 2003; Dumas, Heravizadeh & Hofstede, 2001). In many of these areas, the term service has several common characteristics, for example, functionality, quality, and delivery.

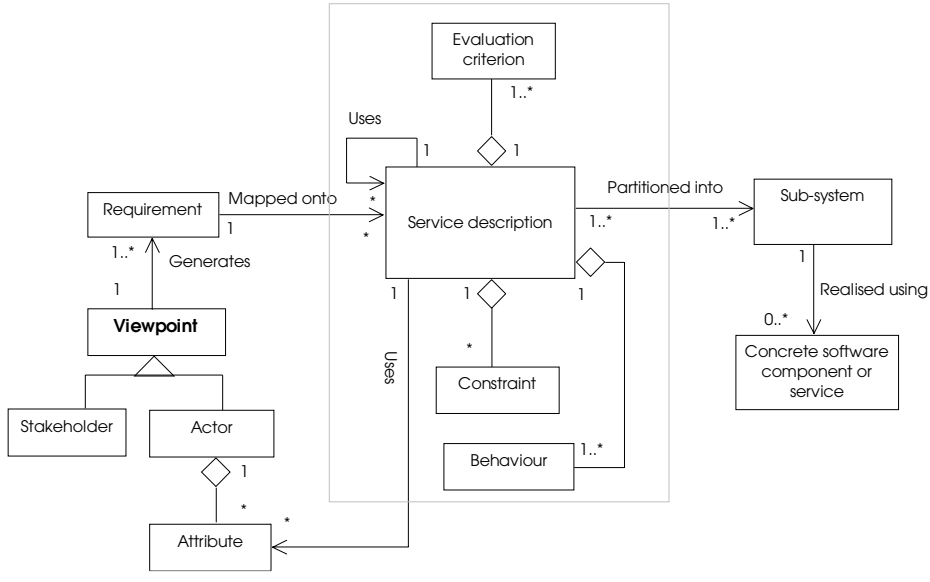
In COMPOSE, a service description is characterized by at least one identifiable function, a trigger by which the service commences, a recipient (Actor viewpoint), service provider (proposed system/component), conditions for service delivery, and constraints on service provision (Figure 3). Service descriptions are derived from viewpoint requirements and represent a level of abstraction in the requirement description. Service descriptions may be partitioned into abstract subsystems at design, which may be realized (composed) using concrete software components or services.

Use cases provide high-level service descriptions (that is, the underlying system functionality). UML sequence diagrams provide detailed service descriptions and capture interactions between services. Sequence diagrams are augmented with state diagrams to capture the system behavior in the context of specific services.

Service descriptions provide a mechanism for modeling viewpoint requirements and for mapping requirements to concrete software components and services.

A service description comprises the following elements:

Figure 3. COMPOSE service model



Invocation: <Set of parameters required by a service and how the parameter values are used by service. Parameters correspond to attributes in the service model>

Behavior: <Specification of the system behavior that results from the invocation of the service. This can be described at different levels of abstraction to aid understanding and component selection>

Constraints: <Description of constraints on service>

Evaluation criteria: <Tests that should be carried out to evaluate a component's conformance with service>

Constraints define the overall qualities or attributes of the resulting system and are derived from nonfunctional requirements. Nonfunctional requirements may constrain the way the system is constructed or the way services are provided. Because they are restrictions or constraints on system services, nonfunctional requirements greatly influence the design solution.

In COMPOSE, service constraints are considered from two different perspectives:

- *Actor Viewpoint*: An actor viewpoint might require that a service be provided with a certain level of quality (for example, availability, response time, format and so forth).
- *Stakeholder viewpoint:* A stakeholder viewpoint might not interact directly with the target system but might express an interest in the overall dependability of the

system, the resources required to provide the services or the conformance of the services with various regulations and standards.

This viewpoint/service-centric approach provides a framework for:

- Reasoning about the events that are responsible for triggering or suspending services. Because the events arise from actors in the system environment, it is possible to specify the required control information from the point of view of the actors. This information is represented as data attributes in viewpoints.
- Structuring service requirements across instances of a viewpoint.
- Integrating functional and nonfunctional requirements. A viewpoint can impose quality constraints on the services it requires from the target system. For example, a viewpoint may require that a service have an *availability* of 98% (say between 8pm – 6pm, Monday to Friday).

A constraint description comprises the following elements:

Identifier: <Constraint identifier>

Type: <Constraint type (for example, availability, response time, format, safety, security, and so forth)>

Rationale: <Justification for constraint>

Specification: <Specification of constraint>

Scope: <Identifiers of services affected by constraint>

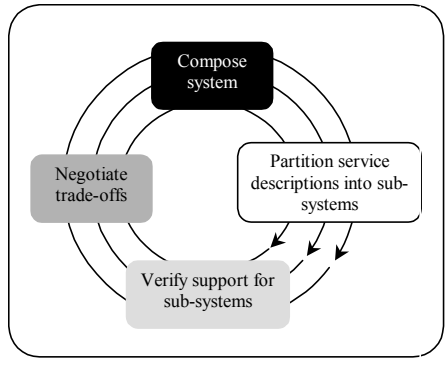
Evaluation criteria: <Tests to evaluate a component's conformance with constraint>

In COMPOSE, certain types of constraint find use at the requirements stage where they are used to support the process verifying software components. Constraints related to effort, vendor and system resource requirements (for example, cost, schedule, hardware, operating system, and standards) might provide the specifier with a mechanism for establishing the availability and viability of a component-based solution. Other types of constraint (for example, dependability) filter down to the design stage where they form the basis for identifying suitable architectures as part of a negotiated design process.

Design and Composition

The main aim of any design process is to achieve *fitness* for use. This is achieved when a set of software components and services have an acceptable match with system context. The system context is defined by the system requirements, cost, schedule, and operating and support environments. Formal Architecture Description Languages (ADLs) have emerged as an effective way of designing and composing component-based

Figure 4. Component-based design process



systems (Medvidovic & Taylor, 2000). In COMPOSE, the design process is driven by a service-oriented ADL (Kotonya et al., 2001).

The design starts with the partitioning of service descriptions into logical subsystems as part of the iterative process shown in Figure 4. In principle, the process of partitioning service descriptions should be straightforward (for example, initial partitioning may be driven solely by architectural considerations). However, there is never a clean match between service descriptions and concrete software components and services; abstract services may therefore have to be reassigned or requirements renegotiated. Partitioning is therefore subject to a negotiation process that must take into account capabilities of available components, business concerns, and viable architectures.

Figure 5 shows how a typical top-down process may be used to partition the system by clustering services to reflect desired architectural and system properties. A security constraint may, for example, give rise to an architecture where security-critical services are held in a single component at the lower levels of layered architecture to ensure a certain level of security. Performance needs may result in a dynamic architecture where popular service components are replicated with increasing load. For these reasons, support for negotiation in the design process is essential. For cases where no suitable off-the-shelf solution can be found (for example, in the case of critical components), the subsystem design may be viewed as a *placeholder* for custom development. This allows the developed subsystem to acquire the pluggability of a component while maintaining consistency with the global system requirements.

Service descriptions focus on system behavior and associated constraints providing a simple but effective mechanism for mapping requirements to component and service architectures. The flexibility and implementation-independent nature of services means the engineer can explore different technologies to compose the system. This may, for example, result in a hybrid system where certain services are provided using black-box components while others are delivered using Web services. There are several reasons why a hybrid solution might be preferred to a purely component or service-based approach:

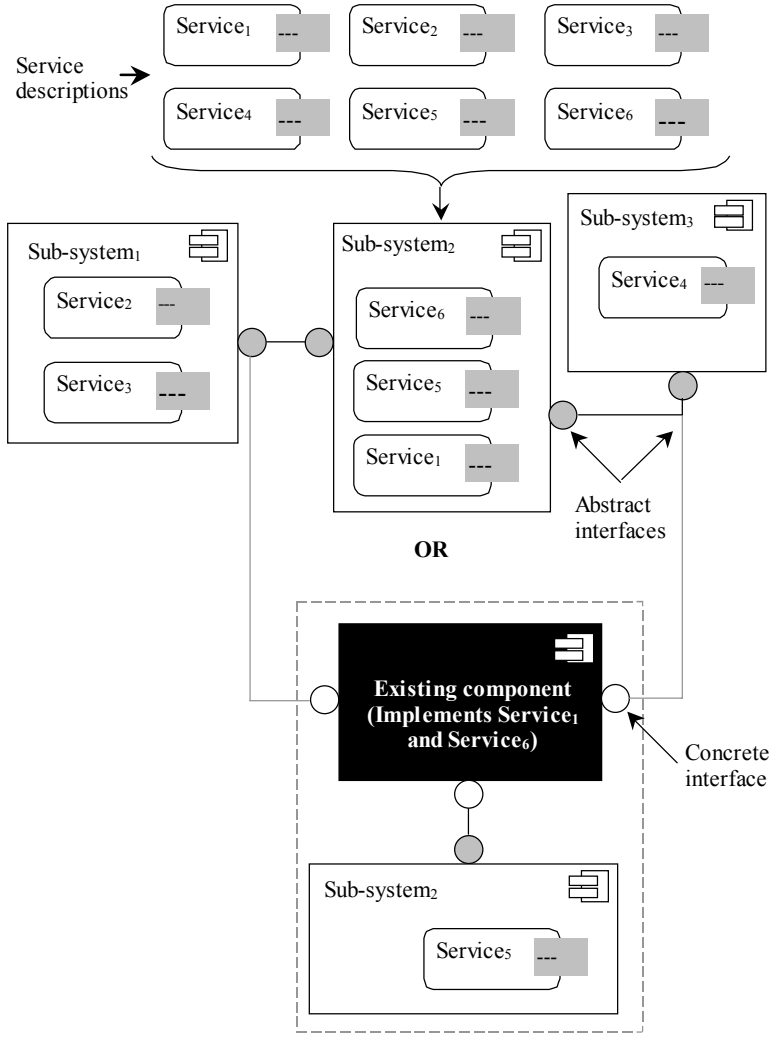
- *Efficiency*: The document-exchange model that is inherent to a service-centric approach is incredibly inefficient. The text-based nature of XML means that the amount of information transferred in bits is much larger than is simply required to encode the information, and there is significant overhead involved in parsing and generating XML documents. This means that a service-centric approach is unsuitable for real-time systems with tight performance deadlines and for the development of systems that process high volumes of transactions.
- *Competitive advantage*: If all organizations move to a development approach where they rely on externally provided services, it is difficult for them to innovate in their IT systems to gain a competitive advantage. Therefore, there will always be a class of companies who are unwilling to be dependent on external suppliers and who will rely on conventional software development.
- *Trust:* While a service-oriented approach may be adopted within an organization or group of cooperating organizations with existing trust relationships, the vision of service brokering and the use of unknown service providers is a very risky one. The associated risks mean that businesses will be reluctant to move to an entirely service-centric approach to software development.
- *Dependability*: Current languages for service orchestration are limited to relatively simple composition constructs and do not provide facilities for exception management. Similarly, current Web service and grid service standards assume that a requested service will be available and will function correctly. Services do not define their failure modes, their limitations, and the quality of service that they offer. Generally, dependability depends on transparency, and the opaqueness of service-centric systems means that it will be difficult to write dependable systems using anything apart from very simple services.
- *Services as an exception-handling mechanism*: For many systems and components, more than 50% of the code in the system is required to handle rare events and exceptions. This introduces significant overhead for *normal* users of the code. By integrating components and services, we have the opportunity to produce much leaner components that are used within a program and to handle exceptions, where performance is less important, using externally provided services.

Verification

Component Selection

This is achieved by formulating selection filters to match requirements to a checklist of component and service properties. Checklist items might range from component functionality, documentation, certification, and vendor support to resource requirements and cost. Table 1 shows an example of a checklist table. The requirement(s) on the top left of the table are matched against the candidate components and services in column 2, using selection filters as follows:

Figure 5. Partitioning services



(1) Select the highest ranked requirement from the list of ranked requirements.

(2) Select a product from the software component/service shortlist (that is, $C_1, C_{2,}..S_1$), where C_i corresponds to a component and S_i to a service.

(3) For each checklist question, determine the extent to which the response is positive in relation to the component/service. Score each response as follows:

- **2**, if the response is positive**.**
- **1**, if the response is weakly positive or there is lack of adequate information to make a definitive judgement.
- **0**, if the response is negative.

(4) Repeat 2–3 until the short listed components/services are exhausted.

(5) Apply appropriate selection filters. Filters are used to focus the selection of components/services on critical properties.

(6) Repeat 1–5 until all ranked requirements are exhausted.

Filters are reusable artifacts with the following structure:

Identifier: <Filter name>

Description: <Description of filter and its effect>

Predicate: <Predicate over checklist questions>

The formulation and selection of filters is likely to be influenced by the nature of the application being assembled and the business concerns of the organization. For example, if we assume that our starting point is a set of components, T_1, such that:

$$T_1 = \{C_1, C_2, C_3, C_4, S_1\}$$

We can define a "fast" filter f_1 such that only members of the set T_1 that support the selected requirement or can be configured to support it are selected. Filter f_1 is defined by the predicate, where *c* represents the general component and *checklist(i)* represents the checklist item i:

$$\forall c: T_1 \bullet (c.checklist(1) \geq 1)$$

T_2 represents the result of applying f_1 to T_1:

$$T_2 = \{C_2, C_3, C_4\}$$

Filters can be combined to provide more a refined selection. Filter f_2:

$$\forall c: T_i \bullet (c.checklist(1) = 2 \vee (c.checklist(1)=1 \wedge c.checklist(7)= 2))$$

may be applied to T_2 to ensure that all components that need to be reconfigured also have help desk available. Thus the set T_2 contracts to set T_3:

$$T_3 = \{C_2, C_4\}$$

By relating development aspects, such as risk and effort to appropriate checklist items, we can formulate filters to minimize their adverse effects. Table 1 shows how the various checklist items relate to effort and risk. Effort corresponds to the time and resources required to realize the feature. Risk corresponds to the probability that providing the feature might cause the project to experience undesirable events, such as cost overruns, schedule delays, or even cancellation. Effort and risk can be categorized as low, medium, and high. The relationships shown in Table 1 are based on the dominant development aspect and are not necessarily exclusive.

Assuming the checklist item scores of *2*, *1*, and *0* to correspond to low, medium, and high risk/effort, we can design risk and effort sensitive filters. The following filter focuses on functionality, component cost, system resource requirements, and component certification to select components and services that reflect low risk and medium to low effort requirements:

$$\forall c: T_i \bullet (c.checklist(1) \geq 1 \wedge c.checklist(2)=2 \wedge c.checklist(4)= 2 \wedge c.checklist(6)= 2 \wedge c.checklist(10) \geq 1 \wedge c.checklist(11) \geq 1)$$

The following filter is an example of a low risk, low effort filter:

$$\forall c: T_i \bullet (c.checklist(1) = 2 \wedge c.checklist(2)=2 \wedge c.checklist(4)= 2 \wedge c.checklist(6)= 2 \wedge c.checklist(10)= 2 \wedge c.checklist(11)= 2)$$

Testing Components and Services

The testing of software components and services is constrained by the lack of source code as well as difficulty in performing *direct* tests in the case of Web services. Therefore, the system integrator is restricted to performing only black-box testing. In the case of Web services, trust schemes might be the only viable means of verification. Onyino et al. (2002) proposes trust model for component-based system development that uses context-sensitive trust variables such as product, project, and business concerns to identify appropriate trust schemes. The model provides a framework for combining various trust schemes (for example, contractual, certification and experience-based schemes).

Detailed testing regimes are out of the scope of this chapter. However, the various ways of expressing services described in the Modeling Requirements section provide the engineer with a good basis for developing black-box test cases (evaluation criteria) for services.

In COMPOSE, verification may involve any or all of the following activities (Rosenblum, 1997):

1. *Component functionality*: The integrator or engineer needs to thoroughly test a new component to verify its functionality prior to deploying it in a larger system.

Table 1. Component preselection using filters

Requirement: 1. Requirement xyz							
Checklist		**Components/Services**					**Related**
Id	**Question**	**C_1**	**C_2**	**C_3**	**C_4**	**S_1**	**Development Aspect**
1	Does component/service support requirement? Yes explicitly = 2; Not explicitly, but can be configured to support requirement = 1, Don't know/does not support feature = 0;	0	2	1	1	0	Effort
2	Is the component/service specification provided? Yes, detailed = 2; Yes, limited = 1; No = 0	2	2	1	2	1	Risk
3	Are release notes provided? Yes, detailed = 2; Yes, limited = 1; No = 0	2	0	1	1	1	Risk
4	Are installation notes provided? Yes, detailed = 2; Yes, limited = 1; No = 0	0	2	1	2	1	Effort, Risk
5	Is component/service available for evaluation? Yes, full functionality = 2; Yes, restricted functionality = 1; No = 0	2	0	2	2	1	Risk
6	Is component/service certified? Yes, independent certification = 2; Yes, local certification =1; No =0;	2	2	1	0	1	Risk
7	Is help desk support available? Yes, continuous = 2; Yes, limited =1; No =0	2	1	0	2	1	Effort, Risk
8	What is vendor's market share? Good =2; Moderate =1; Don't know/Poor =0	2	0	2	1	1	Risk
9	What is maturity of producer development process? CMM Level ≥ 3 = 2; CMM Level 2 =1; CMM Level 1/Don't know = 0	2	1	0	0	1	Risk
10	Are system resources needed by component/service available? Yes =2; Not sure =1; No =0	2	1	2	1	1	Effort
11	Is component/service cost within estimated cost? Yes = 2; No, but acceptable =1; No =0	1	2	1	2	2	Effort

2. *Architectural analysis*: For certain types of systems, architectural analysis may be required to establish how well the design supports desired quality attributes (for example, performance, security, availability, and so forth).
3. *Assembly and integrated system*: If a new component is added to the system or an older version was replaced, the integrated system must be tested. Testing should also be done if the system configuration is altered.
4. *Regression testing*: It is a good idea to perform regression testing on selective critical system components whenever new versions of other constituent components are installed in the system.

5. *Nonfunctional testing*. Various kinds of nonfunctional testing on the system are required to ensure that the system meets the desired level of performance, dependability, stress, and loading.

Negotiation

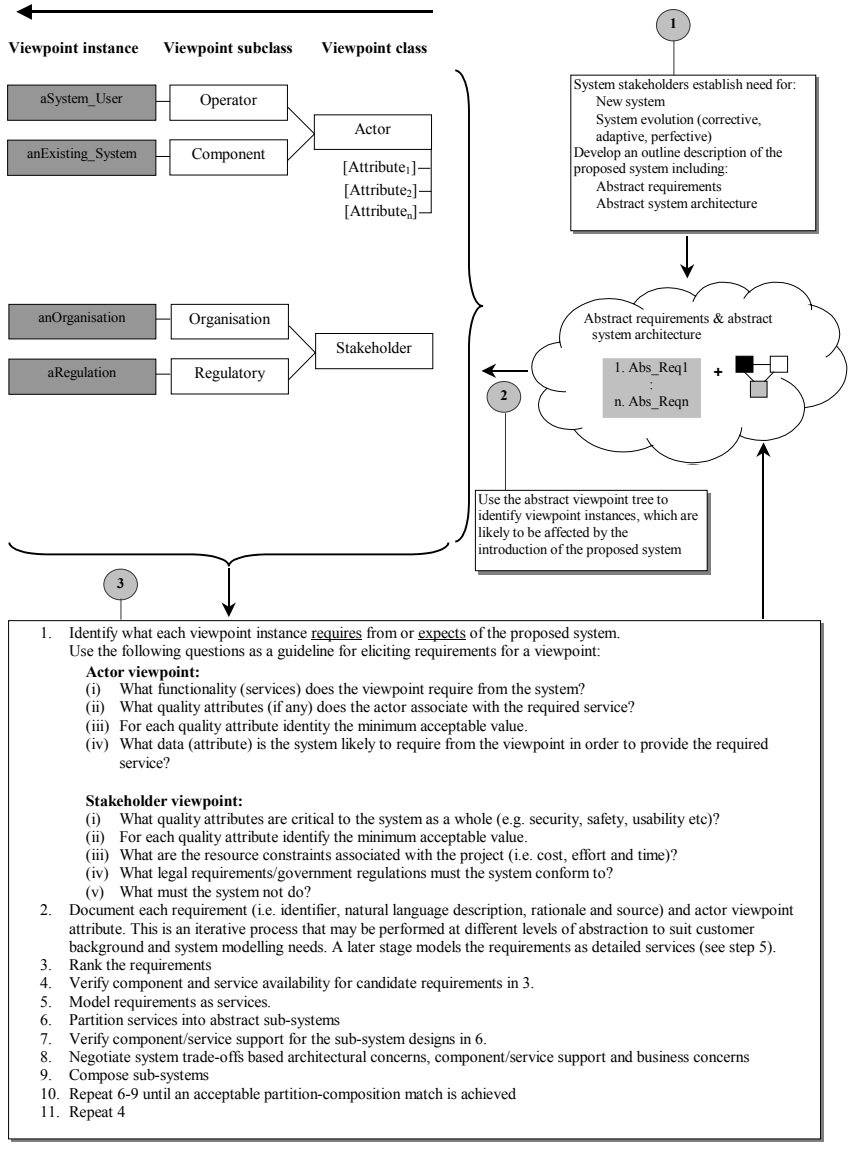
The negotiation process attempts to find an acceptable trade-off amongst multiple (often) competing system attributes. For example, in cases where requirements need to be ranked or where decisions need to be made on alternative designs. The negotiation process described here uses the Simple Multi-Attribute Rating Technique (SMART) to support trade-off analysis. However, other decision support techniques, such as AHP, may also be used. SMART is a powerful and flexible decision-making tool. Because of its simplicity of both responses required of the decision maker and the manner in which these responses are analyzed, SMART has been widely used. The main stages in the SMART analysis are as follows:

1. Identify the decision maker or makers. Examples might include the project managers, customers, system integrators, and system maintainers.
2. Identify the intended goal for the analysis (for example, best design).
3. Identify factors or criteria important in satisfying the goal.
4. Where appropriate, identify subcriteria under each criterion. The lowest level of criteria/subcriteria represents an attribute of the alternatives that can be objectively evaluated.
5. Identify the alternative ways of achieving the goal.
6. Weight the criteria and rate alternatives. Quantify the relationships between criteria by establishing the relevant importance of criteria. Values assigned to criteria are likely to vary with organization and application.
7. Determine how well alternatives score against the lowest criteria. Similarly, the scoring scheme used for alternative is likely to vary with organization and application.
8. For each alternative, compute the weighted average of the values assigned to that alternative.
9. Make a provisional decision.

Method Summary

Figure 6 shows the COMPOSE method steps including requirements definition.

Figure 6. COMPOSE method



Example

We will now illustrate aspects of the method using a subset of requirements extracted from the specification of a real electronic document delivery and interchange system (EDDIS) (Kotonya, 1999). The illustration focuses on requirements and design. The EDDIS runs on a Windows 2000/Windows XP platform and its main function is to manage the process of identifying, locating, ordering, and supplying electronic documents. Users access the system via a Web-based interface using valid usernames and passwords. EDDIS users have access to a range of services determined by the permissions associated with the accounts they use. EDDIS will have a local administrator whose task will be to set up and manage user accounts.

Before an EDDIS user can place a document order, the user must first obtain documents and location identifiers from a centralized document registry. Document orders are placed with the document supplier. All document interchange will use the Z39.50 document retrieval protocol. In this example, we will consider a small but diverse subset of EDDIS requirements. Table 2 shows the EDDIS requirements together with associated viewpoints.

Requirements Modeling and Documentation

Service descriptions can be documented at different levels of abstraction using a variety of notations to aid user understanding and to facilitate mapping to component and service architectures. Figures 7-10 show how uses-cases and state transition diagrams can be used to model and specify abstract services at different levels of abstraction. Figure 8 and 9 show the state diagrams for *user validation* and *document search* services.

Subsystem Interfaces and Relationships

Figure 11 shows a typical service partitioning. The final partitioning is subject to the availability of suitable components and services. In this case the document services, document search, document locate, and document order are provided by black-box components, while the document registry and document supplier services are provided as Web services.Subsystems can be used to model interactions between the services they represent as shown in Figure 12. Subsystems are flexible grouping constructs and the services they represent are likely to be a trade-off between the desired architecture and the available components and services. The interfaces associated with subsystems are a function of the services represented and the external interaction between the subsystem and other subsystems. Figure 13 shows an example of interface identification for the EDDIS system.

Table 2. EDDIS viewpoints and requirements

Viewpoint			Requirement			
Type	**ID**	**Role**	**ID**	**Description**	**Rationale**	**Ranking**
Operator	1	EDDIS_user	1.1	EDDIS users shall be able to login onto the system via a Web-based interface using valid usernames and passwords.	To provide a universal access to EDDIS services	Essential
			1.1.1	Once logged in, EDDIS users will have access to a set of services determined by the permissions associated with their accounts.	To provide a simple mechanism for managing user accounts.	Important
			1.2	EDDIS shall allow users to search for and identify documents, which interest them. A document search will be initiated by a search criterion and a list of databases to be searched. The output will be a set of document identifiers.	Basic EDDIS functionality	Essential
			1.3	EDDIS shall allow users to determine the location of documents. A document locate service will be initiated by a set of document identifiers and the output shall be a set of location identifiers.	Basic EDDIS functionality	Essential
			1.4	EDDIS user shall allow users to order documents. A document order will be initiated by a set of document and location identifiers. The output will be a set of order identifiers and electronic documents.	Basic EDDIS functionality	Important
Operator	2	EDDIS_ administrator	2.1	EDDIS shall provide facilities for setting up and managing user accounts.	Basic EDDIS functionality	Important
Component	3	Document_ supplier		The document order client will use the Z39.50 document retrieval standard.	Document retrieval standard used by document suppliers	Essential
Component	4	Document_ registry	4.1	EDDIS shall be able to access a centralized document registry to obtain document and location identifiers using the Z39.50 document retrieval standard.	Document retrieval standard used in document registry	Important
Stakeholder	5	EDDIS_ consortium	5.1	The system shall run on Microsoft Windows 2000 and Windows XP	Most users are likely to use a Windows-based PC to access EDDIS services	Important
			5.2	The system shall ensure that a reasonable level of performance is maintained across the services at all times.		Useful

Conclusion

This chapter has identified the challenges and problems likely to be faced by organizations intending to adopt component and service-oriented software development. While component and service-oriented development offer significant advantages over traditional development approaches, they also carry significant risk throughout the system life cycle. Part of this risk is related to the lack of effective processes and methods that

Figure 7. Using use cases to model EDDIS services

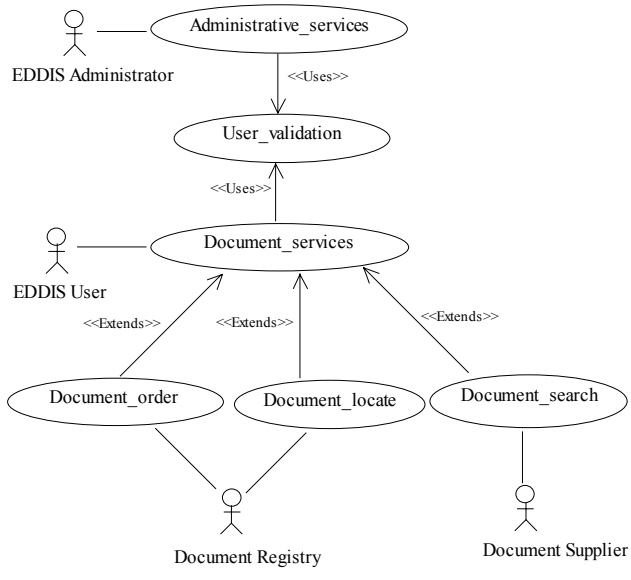


Figure 8. Use case specifications

Use case specification

Name: Document_services
Uses: User_validation
Extends:
Participating actors:
EDDIS_User
Entry conditions:
1. Valid username
2. Valid password

Flow of events:
1. EDDIS user enters a username and password
2. If username and password are valid:
   2.1 System initializes user account permissions
   2.2 Displays the services available to the user
   else
   2.3 System prompts the user to re-enter username and password

Exit conditions:
1 System resets user account permission
2 Closes user account

Constraints:
1 The service shall be available on Microsoft 2000/XP platform
2 Service shall have a reasonable level of performance at all times

Use case specification

Name: Document_search
Uses:
Extends: Document_services
Participating actors:
EDDIS_User, Document Registry
Entry conditions:
1 Document_search $\in$ available_services
2 Document databases $\subseteq$ set of user permissible databases

Flow of events:
1 EDDIS user enters search criterion and a set of document databases
2 If document is found a set of document identifiers is displayed else a "document not found" message is displayed
3 Search criterion is retained in user workspace for future searches

Exit conditions:
1 Service access conditions are reset

Constraints:
1 Service conform to Z39.50 document retrieval standard

Figure 9. User-validation and document services state diagram

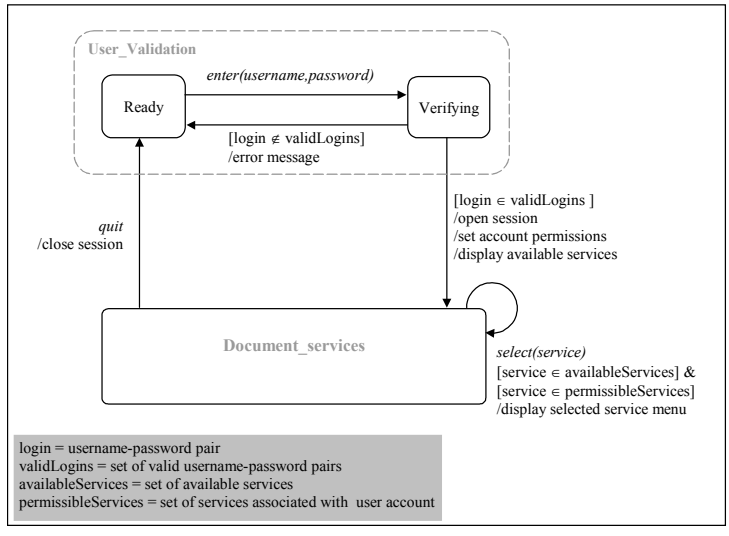


Figure 10. State model for Document_search service

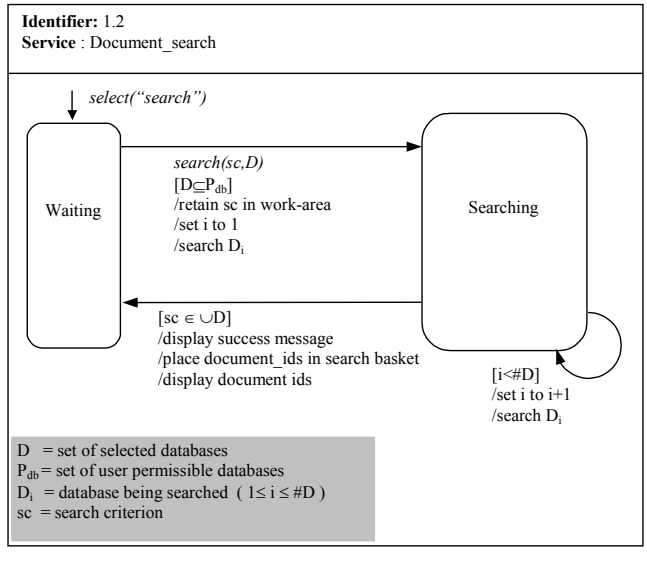


Figure 11. Service partitioning

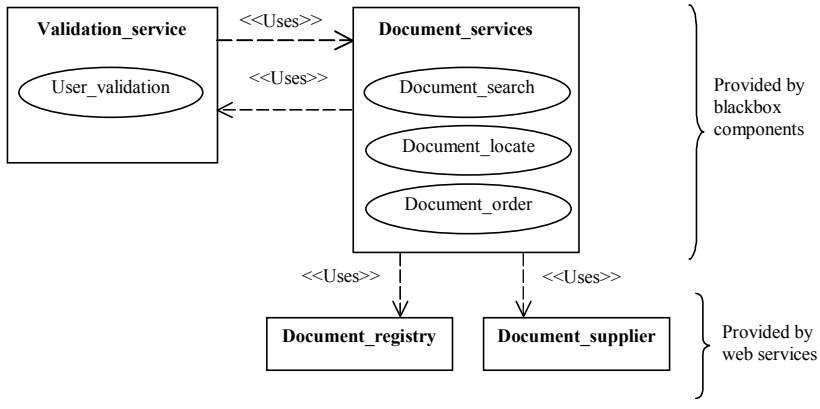


Figure 12. Service subsystem interaction diagram

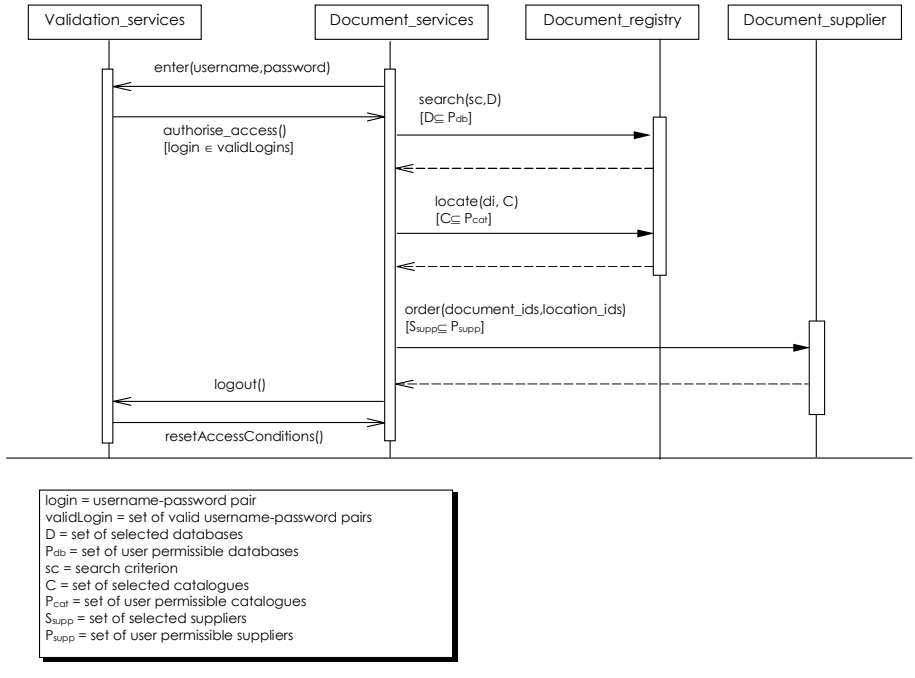
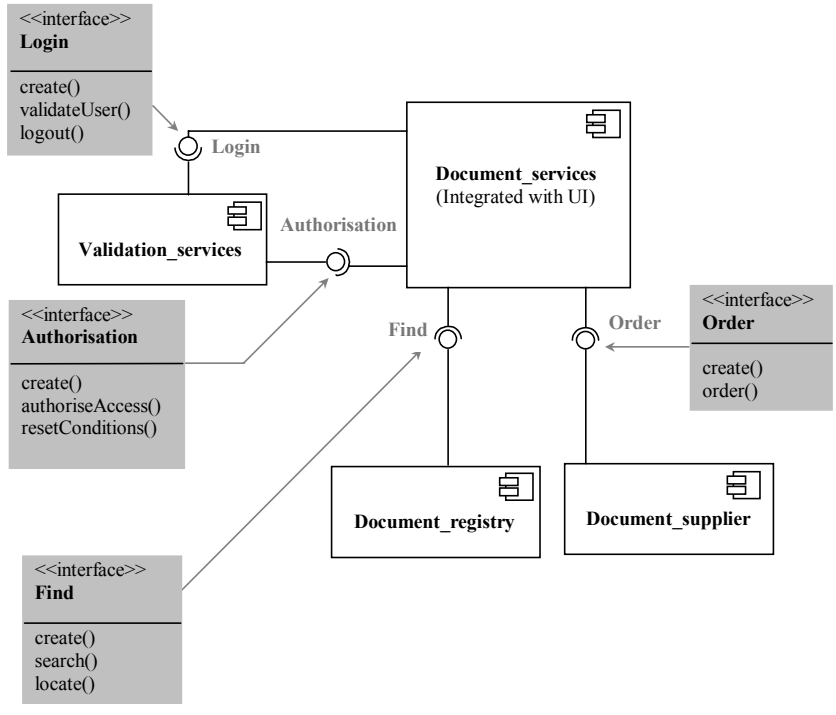


Figure 13. Interfaces identification



support component and service-oriented development. Component and service-oriented development are highly iterative processes requiring simultaneous consideration of the system context (system characteristics, such as requirements, cost, schedule, operating and support environments), capabilities of the software component or service, the marketplace, and viable system designs. The diverse nature of software systems also means that it is unlikely that systems will be developed using a purely service or component-based approach. Rather, a hybrid model of software development where components and services coexist in the same system is likely to emerge. There is a general lack of methods that support this type of hybrid development. Our solution has been to develop COMPOSE as a service-based, negotiation-driven approach that supports a hybrid component/service-oriented development. COMPOSE provides a framework for integrating different methods and techniques and for mapping these to a generic development with reuse process. COMPOSE provides an intuitive scheme for eliciting and modeling requirements and for mapping these to component and service architectures. It also provides the developer with a pluggable basis for custom development in cases where available components or services are inadequate or inappropriate. The process is supported by verification and negotiation at different levels of abstraction.

Acknowledgments

The work described in this chapter is being undertaken as part of the project ECO-ADM (IST 20771) funded under the EU IST Program. We are grateful for the contribution of our partners in ECO-ADM: CCS, Ingegneria Informatica, EIDOS Sistemi Di Formazione and DHL, Ireland.

References

Arsanjani, A., Hailpern, B., Martin, J. & Tarr, P.(2003). Web services: Promises and compromises. *ACM Queue*, 48-58.

Bennet, K.H. & Gold, N. (2003). Achieving Ultra Rapid Evolution Using Service-based Software. *Proceedings of the 4th International Workshop on Principles of Software Evolution, European Software Engineering Conference*, Helsinki, Finland

Boehm, B. & Abts, C. (1999). Integration: Plug and pray. *IEEE Computer,* 32(1), 135-138.

Brown, A.W. & Wallnau, K.C. (1998). The current state of CBSE. *IEEE Software*, 15(5).

Cerami, E. (2002). *Web service essentials*. O'Reilly & Associates.

Crnkovic, I., Hnich, B., Jonsson, T. & Kiziltan, Z. (2002). Specification, implementation and deployment of components: Clarifying common terminology and exploring component-based relationships. *Communications of the ACM,* 45(10), 35-40.

Dumas, M., Heravizadeh, J. & Hofstede, D. (2001, April). Towards a semantic framework for service description. *Proceedings of the International Conference on Database Semantics*, Hong Kong.

Karlsson, J., & Ryan, K. (1997). A cost-value approach for prioritising requirements. *IEEE Software*, 14 (5), 67-80.

Kim, S.D. (2002). Lessons learned from a nationwide CBD promotion project. *Communications of the ACM*, 45(10), 83-87.

Kotonya, G. (1999). Experience with viewpoint-based specification. *Requirements engineering*, 4(3), 115-133.

Kotonya, G., Hutchinson, J., Onyino, W. & Sawyer, P. (2002, April). Component-oriented requirements expression. *Proceedings of the 16th European Meeting on Cybernetics and Systems Research*, Vienna, Austria.

Kotonya, G., Onyino, W., Hutchinson, J. & Sawyer, P. (2001). *Component architecture description language (CADL)*. Technical Report, CSEG/57/2001, Computing Department, Lancaster University.

Kotonya, G. & Rashid, A. (2001, December). A strategy for managing risk in component-based systems. *Proceedings of the 26th IEEE Euromicro Conference*, Warsaw, Poland.

Kotonya, G., Sommerville, I., & Hall, S. (2003, September). Towards a classification model for CBSE research. *Proceedings of the 29th Euromicro Conference*, Antalya, Turkey.

Layzell, P.J., Bennett, K.H., Budgen, D., Brereton, O.P., Macaulay, L.A. & Munro, M. (2000, December). Service-based software: The future for flexible software. *Proceedings of the Asia-Pacific Software Engineering Conference*, Singapore.

Lootsma, F.A. (1999). *Multi-criteria decision analysis via ratio and difference judgement*. Kluwer Academic Publishers.

Medvidovic, N. & Taylor, R.N. (2000). A classication and comparison framework for software architecture description languages. *IEEE Transactions of Software Engineering*, 26 (1), 70-93.

Nadhan, E.G. (2003). Service oriented architecture implementation challenges. *EDS.com* [Online]. Available: *http://www.eds.com/thought/thought_leadership_so_architecture.pdf*

Ncube, C. & Maiden, N (1999). PORE: Procurement-oriented requirements engineering method for the component-based systems engineering development paradigm. *Proceedings of the 2nd IEEE International Workshop on Component-Based Software Engineering*, Los Angeles, California, USA (May 1-12).

Onyino W., Kotonya, G., Hutchinson J., & Sawyer, P. (2000, June). Towards an inclusive model of trust for COTS-based software development. *Proceedings of the International Conference on Software Engineering Research and Practice*, Las Vegas, USA.

Rosenblum, D.S. (1997). *Adequate testing of component-based software.* Technical Report No. 97-34. University of California, Irvine.

Saaty, T.L. (1980). *The analytic hierarchy process*. New York: McGraw-Hill.

Stal, M. (2002). Web services: Beyond component-based computing. *Communications of ACM*, 45(10), 71-76.

Szyperski, C. (2002). *Component software: Beyond object-oriented programming (2nd edition)*. Addison-Wesley.

Szyperski, C. (2001, January). *Component and Web services*. Software Development Media.

Vigder, M., Gentleman, M. & Dean, J. (1996). *COTS software integration: State of the art*. Institute for Information Technology, National Research Council, Canada.

Chapter IX

Architecture, Specification, and Design of Service-Oriented Systems

Jaroslav Král
Charles University, Czech Republic

Michal Žemlička
Charles University, Czech Republic

Abstract

Service-oriented software systems (SOSS) are becoming the leading paradigm of software engineering. The crucial elements of the requirements specification of SOSSs are discussed as well as the relation between the requirements specification and the architecture of SOSS. It is preferable to understand service orientation not to be limited to Web services and Internet only. It is shown that there are several variants of SOSS having different application domains, different user properties, different development processes, and different software engineering properties. The conditions implying advantageous user properties of SOSS are presented. The conditions are user-oriented interfaces of services, the application of peer-to-peer philosophy, and the combination of different technologies of communication between services (seemingly the obsolete

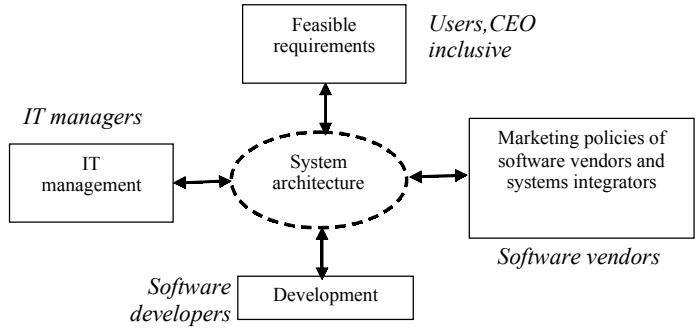
ones inclusive), and autonomy of the services. These conditions imply excellent software engineering properties of SOSSs as well. Service orientation promises to open the way to the software as a proper engineering product.

Introduction

Service orientation (SO) is becoming the central topic of software engineering. There is an explosive growth in the number of conferences, products, and articles discussing and using the principles of SO and service-oriented architectures (SOA). Service-oriented software systems (SOSS) are of different types depending on the character of the functions the system provides, the system environment (for example, e-commerce or a decentralized international enterprise), and the way the system is developed. The common property of SOSS is that their components behave like the services in real life mass service systems. The SOSS must then be virtual peer-to-peer (p2p) networks of autonomous components (services). The services can have various properties; they need not be Web services in the sense of W3C (2002) and need not therefore use standard communication protocols, compare Barry and Associates (2003) and Datz (2004).

We shall show that the software engineering properties as well as the user-oriented properties of any SOSS strongly depend on the properties of the service interfaces and that user interfaces of the system should be implemented as specific services (peers of the network) as well. All these issues are related to the architecture of the system. We will discuss how the properties of the architecture influence the set of feasible functions, development (especially the requirements specifications), feasible development techniques (for example, agile ones), standards, politics of IT management, and marketing strategies of software vendors and/or system integrators (Figure 1). The feasible functions of SOSSs include the functions important for user top-management.

Figure 1. Central role of system architecture



Feasible functions of any large system depend on its architecture. The decision as to what architecture is to be used must therefore be formulated in early stages of the system life cycle. On the other hand, the structure, techniques, and content of requirements specifications are influenced by the properties of the system architecture and the details of its implementation. We shall show that SOSS should use a combination of various techniques developed during the software history (for example, message passing, object orientation, common databases, and, sometimes, batch-oriented systems). All these issues should be addressed in the specifications of SOSSs. SO is a paradigm new for many software people. It implies some problems with the application of SO.

Peer-to-Peer Information Systems (P2PIS)

Large information systems must often be developed as a network of loosely coupled autonomous components — services (possibly information systems) integrated using peer-to-peer principle (further P2PIS). The change of the architecture should be accompanied by changes in requirements specification that should reflect the service-oriented structure of the system.

The specification of P2PIS starts from the specification of system user interface (portal) and from the specifications of the services. The specification of services starts from the definition of their interfaces. It can be accompanied by specification of the services of the infrastructure (message formats, communication protocols, middleware services, in general). Services in P2PIS can be newly developed applications, encapsulated legacy systems, or third party products. P2PIS enables new types of requirements (for example, the requirement that a P2PIS should support decentralized and flexible organization of a global enterprise, see Král & Žemlička, 2003) and makes achievable software engineering properties like reusability, flexibility, openness, maintainability, the use of legacy systems and third party products, or the reduction of development costs and duration. Experience shows that such systems can be extremely stable (Král, 1995).

There are two main variants of P2PIS. The first one is used in e-commerce where the service starting a communication must first look for communication partners. The partners must offer their interfaces (typically specified by WSDL). This schema implies the use of Internet and international standards like SOAP. We shall call such systems *(software) alliances*.

The systems formed by stable sets of services knowing their permanent communication partners will be called *(software) confederations*. Confederations occur often. Examples are:

- Information systems of international enterprises having the structure of a network of autonomous organizational units (divisions). The information systems are formed by a peer-to-peer network of the information systems of the divisions and by some additional components serving the whole enterprise (for example, portals).

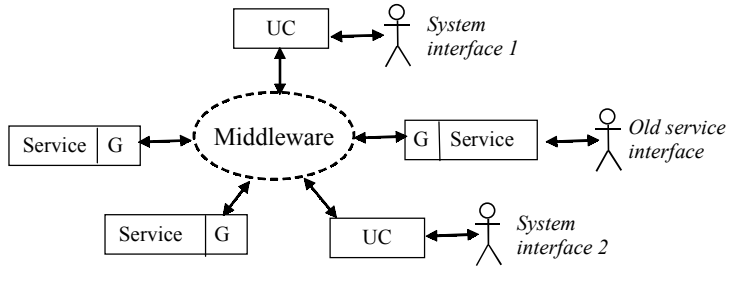
Such an architecture simplifies the integration of new divisions and/or of newly purchased enterprises as well as the selling out or outsourcing of some divisions or splitting the enterprise into smaller ones.

- Information systems of e-government built as a network of the information systems of particular offices[1] (Král & Žemlička, 2001).
- A long-term collaboration between the information system of an enterprise and the information systems of its business partners needed for supply chain management (SCM) (Lowson, King & Hunter, 1999) and customer relationship management (CRM) (Dyché, 2002).
- An open association of health organizations (physicians, hospitals, laboratories, health database services, and so forth) forming an information system intended to simplify, enhance, and speed up health care.
- Process control systems (soft real-time systems) supporting, e.g. computer integrated manufacturing. Such systems were the first systems having main properties of service-oriented systems. They have proved for the first time the advantages of service orientation.

If a system *S* has p2p architecture, it must have structure allowing its peers/services to collaborate. The services must be equipped by gates connecting them to a middleware. The system *S* must usually be equipped by a user interface (portal). There can be several portals. Alliances need not have any portals (Figure 2).

The properties of P2PIS depend substantially on the properties of the interfaces provided by the gates and by the functions of the middleware. The most important property of the interfaces is how much they vary. Stable interfaces increase the stability of P2PIS, reduce the development and maintenance costs, and hide the implementation details and philosophy of the component. It is shown below that the gates need not transform components into Web services and that the middleware in confederations need not be Internet based. On the other hand, Web services and Internet-oriented middleware are necessary in alliances as the use of worldwide standards and tools is the precondition of e-commerce and of the communication between partners unknown to each other before the communication starts.

Figure 2. Architecture of a service oriented system (G is a gate, UC is an user interface service (portal))



Note that SOSSs are usually built starting from the specification of the interfaces and that the components providing the services are mainly integrated as black boxes.

Alliances

A crucial issue of the development of alliances is the standardization of communication protocols. From the technical point of view, the dialog between partners (peers) in a P2PIS can easily be fully automated, but the business agreements should be (and, due to business and legislative reasons, usually are) controlled (supervised) personally by representatives of the communicating parties.

The dialog is driven by a businessperson (initiator) searching for business partners. The businessperson applies knowledge about the history of collaboration with the partners and about the current situation (that is, credibility) on the market. The partner should evaluate the enterprise of the initiator similarly. The partners have to check whether to conclude a contract. Supervision is necessary as someone must be responsible for any business operations. The business documents produced in this way are often to be formally confirmed by people. It holds for P2PIS in general, but it is especially needed in alliances.

Current practice is to establish the cooperation in alliances via Web services in the sense of W3C. The dialog of business partners then starts using UDDI, WSDL standards, and continues in SOAP.

The coordination of the business processes of several partners can be a complicated task. Optimal solution of the supervision of the stages of business processes is a cornerstone of alliance requirements specification. It is to some degree true for confederations as well. The business processes are often developed using development tools like .NET or J2EE. There are discussions about what choice is the best (see the discussion on e-services in *Communications of the ACM*, July 2003, pp. 24-69).

The main advantage of alliances is their flexibility, generality, and standardized solutions. The disadvantage is the problems with efficiency, stability, the size of the used standards, and problems with integration of legacy systems and third party products.

A deeper problem is that SOAP is not too user friendly as it is based on remote procedure calls (RPC) close to programmers' knowledge domain (for example, object orientation) and not to the user problem domains. It enlarges the problems with requirements specification via WSDL and UDDI. UDDI is a centralized service. Centralized services are not good in p2p frameworks. It is confirmed by the experiences with UDDI systems. SOAP, like object-oriented languages, requires many method calls and, therefore, also many messages during even very simple dialogs. It causes efficiency problems, problems with prototyping and understanding the communication by human beings.

The systems using RPC (for example, SOAP) are better suited to the business operative than to business process analysis, usually based on a common data tier.

Alliances are suited to operative tasks in the global market. Important decisions should often be, however, preceded by analysis of the market and the history of cooperation with

a given partner. It implies user involvement, massive data analysis, and the tools specific for it. The tools often do not fit into RPC/SOAP frameworks.

A very important advantage of alliances is that communicating software parties can be in the framework of the SOAP/Web services developed individually like programs serving, for example, terminals.

It can happen that a business case (process) fails for some reason. In this case, the reasons of the failure and people responsible for it can be detected via the analysis of the messages logged during the corresponding business process. The analysis can be used as evidence at court. The messages stored in a log memory must therefore be understandable for users and experts in economy and even for lawyers, who should be user-oriented. It is not clear whether messages in SOAP format can fulfill this requirement. It indicates that a proper message presentation tier enhancing communication legibility should be available.

Software Confederations

E-Government: Confederation via Integration

We shall demonstrate some typical software confederation (SWC) related issues on the example of e-government. The engine of e-government — state information system, SIS — is one of the largest software systems to be built in any country. Let us now give the list of the most important requirements on SIS:

- SIS should service citizens, enterprises, and state and municipal offices. SIS should be able to communicate with information systems of private companies and/or (potentially) of citizens. To fulfill it, SIS must have a complex subsystem (portal) providing the interface for citizens. Such an interface should be flexible in its functionality depending on the rights/profiles of specific groups of citizens and/or state officers/clerks. There should be one or more user interface gates (portals) providing an integrated interface making the internal structure of the system invisible (transparent).
- SIS should support the collaboration of all state offices and majorities. Examples are the collaboration during the investigation of car robberies and/or document verification.
- SIS should reflect the frequent changes in laws and in the structure of state administration.
- SIS should use autonomous tools, often third-party products for data filtering, mining, and analyzing. It is likely that many new tools will be added in the future.

As there are many systems used by individual offices, it is very difficult to rewrite them in time. The existing system should therefore be integrated without any *substantial* change of its functions. The systems must be easily integrated into SIS without any

substantial reconstruction. There is yet another, maybe substantially important, reason for these requirements. No office will take any responsibility for a (sub)system if there is any doubt it works correctly — there must be the *feeling of ownership* of the (sub)system. It can reduce the resistance of users and/or politicians caused by their apprehensions about their positions and power.

As it is highly desirable that the IS of various offices should be at the local level (in a particular office), used without substantial changes (for example, *business as before*). It usually implies that the interfaces of constituent autonomous information systems (autonomous components/services) tend to be user knowledge domain oriented and coarse grained. We shall see that it offers substantial software engineering advantages as well as many benefits for users. A properly specified and designed software confederation increases the dynamics of the system structure and openness of the system.

The conclusions can be illustrated on the following example. People responsible for SIS of the Czech Republic wanted to redevelop the SIS as a monolithic system from scratch. Practical experiences induced them to accept that the SIS must be a P2PIS.

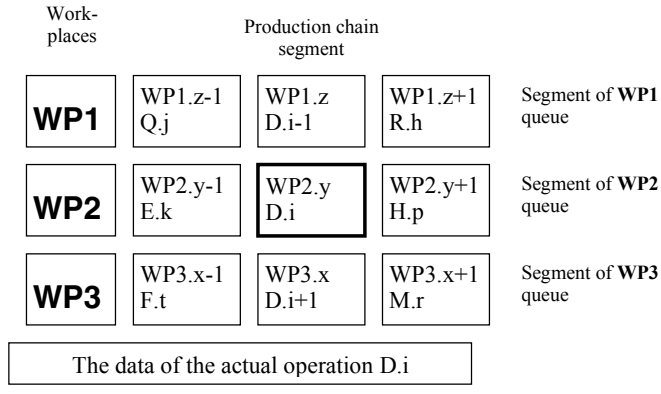
The number of peers in confederations is not too large (compared with e-commerce) and the peers are known. The collection of peers (services) does not vary too quickly. The communication protocols between the peers can then be suited to particular needs; they can be based on nonstandardized tools/solutions without any substantial penalty (compare to Demetriades, 2003). It allows use of various turns known from the history of computing for specific tasks like data reconstruction, data- or object-oriented design for the development of peers.

Manufacturing System: Decomposition and Integration

Systems supporting e-government are the systems developed mainly via integration of existing systems, possibly equipped by appropriate gates and transformed so that they can work as services (peers in a p2p system). Some SOSSs are, however, developed from scratch via decomposition of the system into services. Then the services, user interfaces, and middleware are developed and integrated. It is typical for (soft) real-time systems, for example, in manufacturing. Such systems have shown many advantages of service orientation.

Figure 3 shows the interface of the manager of a flexible manufacturing system producing parts of machine tools (Král, 1995). The manufacturing of the parts is defined by linear sequences of manufacturing operations. A generalization to more complex workflows (for example, assembling) is possible but the interface becomes more complex. The workshop manager chooses the central (actual) operation D.i, and the system shows the previous and next operations in the technological sequence D and in the workplace queues. Note that the manager felt the interface as a support for the standard management activities that were familiar. The manager could add/modify the technological sequence and rearrange the queues. The required data could also be filled by a scheduler from the enterprise level. If the scheduler produced right data, no actions from the manager were needed. We call such types of business processes reconstruction (BPR) the soft one. It should be used as often as possible.

Figure 3. Interface of the manager of the manufacturing system



The interface of the manager was data-oriented — generated from a database. Other parts of the system communicated via commands (for example, store product P in warehouse place WP), so different attitudes (not only RPC) had to be used. It was important that the structure of software services reflected the structure (and interfaces) of real-life services.

These properties of the system were the reasons why the flexible manufacturing system (FMS), an island of automation, was very successfully used for more than twenty years in an enterprise having several successive enterprise information systems. FMS was used without any substantial changes and almost without any maintenance. It was thanks to the fact that the interface has corresponded to the manager intuition, used his knowledge, and supported his skills. To generalize, EAI (service interface) should support the intuition and knowledge of users and should be user and problem oriented. The user should have a chance to influence the design of the interface. User orientation of interfaces offers the possibility to simulate (even substitute) the services (components) not yet existing by communication via portals. We say then that such interfaces are *user performable*. It substantially enhances prototyping tools.

Middleware Enhancement in Confederations

The decision whether to use standard or proprietary message formats should be based on a proper (service-oriented) analysis of the partnerships of autonomous services. The standards in their current form are difficult to use. It can be reasonable for the dynamic enterprises to choose a proprietary solution of message formats. It need not be too difficult to adapt the proprietary solutions to future stabilized standards using the tools like XSLT and PHP. Using the tools like XSLT and PHP, we can build new types of services called *front-end gate* (FEG). FEG is a service used as a front-end part of the gate G of a service S (Figures 4 and 5) or as a router. FEG transforms the formats of input and output messages of S into forms acceptable by the partners of S and hides undesirable properties of the gate G, like disclosing the implementation details of S. The problem is that,

according to our experiments, XSLT is awfully ineffective and unstable today for more complex tasks.

Our two examples show the main extreme situations in the confederation development. In e-government, the system is mainly built via integration of existing systems. In this case, we can define the interfaces only. In the case of manufacturing, the integration stage is preceded by decomposition. In this case, we must specify the *boundaries* of services. They should be intuitively "similar" to the boundaries of real world services.

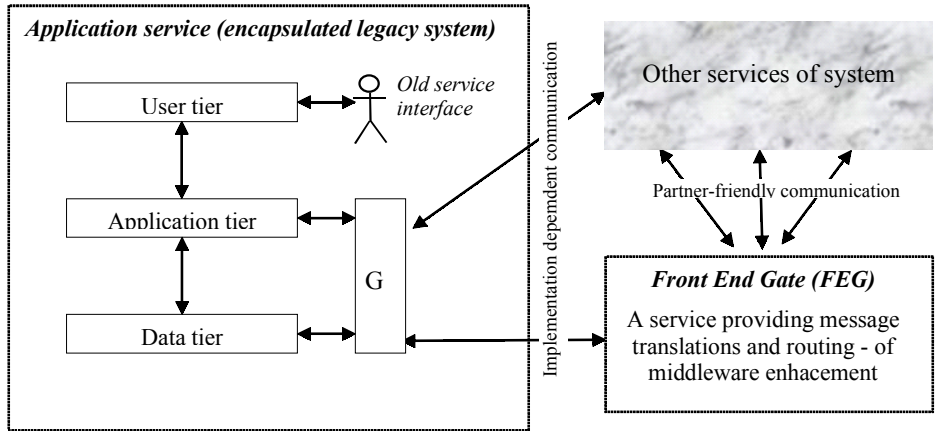
A specific issue in confederations is that we must often also apply attitudes/philosophies known from the software development history. Middleware can be implemented via Internet or via a common database with triggers. CORBA can be used in some situations as well. All the attitudes must be in practice combined, depending on the type of the functionality (for example, commands on operational level and data analysis on management level), so different philosophies can and must be applied and combined in confederations. It increases the complexity of development.

Business Processes in Confederations

Business processes in confederations (BPC) must be composed from steps — functions/actions of peers offered by their interfaces. We have seen that business services must usually be supervised by users — the owners of the processes — or there should be such a possibility. It is also necessary to offer users a tool to define business processes. Both requirements imply that the languages understood by the gates should be based on user knowledge domain languages and notions. They should be *user-oriented*.

The data defining business processes are best to store in system user interface (portals) and the business steps are supervised via a portal or by a temporary service S controlling

Figure 4. Three-tier service (encapsulated information system) with gate G and its communication links



the particular process. An extreme solution is that there are no business processes defining data. Business data analysis (via, for example, OLAP) must be based on a data-oriented view. In this case, the users (usually managers) must have a transparent access to data tiers of the components. It is a very complicated problem (Lenzerini, 2001).

The notions and languages specifying the actions of components (for example, bookkeeping) in a user-oriented way are remarkably stable. The same can therefore hold for the languages of the component interfaces. It is then possible that the interface of a component need not vary if the component implementation varies. The stability of interfaces is important for the communication partners of the given component and for the stability of SOSSs. It has further substantial software engineering advantages (modifiability, outsourcing opportunities, reduction of number of messages, integration of products of various vendors, openness, and so forth).

Historical Technologies Used in Service-Oriented Systems

As mentioned above, it is not feasible to limit the philosophy and the software architecture to one choice, so the crucial point of the requirements specification is what philosophy or combination of philosophies is to be used. This issue is not addressed in existing CASE tools and only a little by research. We shall attempt to find some criteria for the choice of philosophy using historical knowledge.

Due to an improper, too concise coding of date, almost all programs written in COBOL had to be rewritten during the 1990s. It is known as the Y2K problem. It appeared that many enterprises used COBOL programs for years without having any COBOL programmer. Such programs had to be very stable as they were used without any maintenance. Systems written in COBOL are like SOSSs designed as virtual networks of COBOL applications. The communication between the applications is implemented via data stores/files. If input data stores of an application A are ready, A can be started and it works autonomously until input data stores are processed. Thanks to this property, COBOL systems can be developed incrementally provided that input data stores can be easily generated. Each application can be specified as a data transformer and programmed and tested autonomously. It led to specific techniques. COBOL systems usually contained various input filters, data format transformers, report generators, and so forth.

The ability of components to be developed autonomously is the crucial qualitative property of the components of large systems. Batch COBOL systems must often be combined with services of confederations. An example is massive data reconstruction.

Data-oriented systems (DOS) appeared after the success of modern database systems. It was common at that time that the main aim was to computerize operative levels of enterprises (warehouses, bookkeeping, business data). Data types were known and various known operations could be defined over the same data. Moreover, properly designed data structures can be used for many other potential operations — that is, the data enabled many further operations not specified yet. It was and still is supported by

the power of SQL language. The data orientation must be applied in confederations when the services are intelligent, especially if management activities must be supported. Technically, it implies data analysis and presentation tools (Figure 3). There are, however, problems with common database schema and data replication (Lenzerini, 2001). Elements of DOS must be applied in subsystems providing online/interactive data analysis (for example, OLAP/ROLAP).

Implementation Issues

Integration of Legacy Systems

Technically, a legacy system *LS* is integrated into the system via adding a gate *G* to *LS*. *LS* then becomes an autonomous service *AC* (Figure 4). *G* connects *AC* to a middleware. *AC* is also reconstructed to be a permanent process if needed. The software confederation then has the structure from Figure 2. We will often say that *AC* is an autonomous component if we want to express its implementation character.

Although the choice of the p2p architecture seems to be a technical matter, it should be included into requirements specification as it substantially influences user properties of the system and the structure of the specifications. If an autonomous component (encapsulated legacy system) *AC belongs* to some department/division (for example, its local information system), the department is satisfied with *AC*, and *AC* can provide all required information/services to other collaborating autonomous components, then *AC* can and often should be integrated without any substantial reconstruction. It reduces coding effort as substantial parts of the system are reused. The people using *AC* feel that they still *own* it. It is well known that a feeling of ownership is advantageous and often necessary.

Confederative architecture can support new requirement types raised by CEO, like the transformation of the enterprise organization structure into a decentralized form, selective outsourcing of some parts of the information system, selling out some divisions/departments, integration of newly purchased firms/divisions, support for supply chain management, and forming various business coordination groups (Král & Žemlička, 2002).

Front-End Gates

Requirements specifications of software confederations must be based on the properties of the interfaces of autonomous services/components (AC). The AC is used as *black box*, that is, its interface is known only to its partners. This is very important, as interfaces, especially the user-oriented ones, are usually more stable than the implementations (Král & Žemlička, 2003).

Interfaces should hide the implementation details and philosophy of autonomous services. If we, however, need the gate *G* from Figure 5 to offer an access to all

functionality of the corresponding component, the gate must usually disclose the main features of the implementation. So, under such a condition, the message formats of *G* must be, to some degree, implementation oriented. As such, it is not stable enough or optimal for some (usually many) partners of AC. We have seen that we can use *front-end gates* (FEG) to solve the problem. FEG is again an autonomous service (peer) with the properties similar to user components. FEG must usually be developed — it is a white box.

Front-end gates play several roles. They stabilize and generalize the interfaces of components and make them partner-friendly. They can provide several different interfaces of a given AC when different groups of its partners need different interfaces. Different FEGs can provide different security levels or different middleware services for different partners. In some situations, one FEG can provide a parallel access to more than one application component. The condition is that some application components offer similar or complementary functions (for example, in a lecture/test reservation system of a university, different application components may handle different faculties or subject groups).

FEG can direct messages to the components that are less loaded at a given moment. Some services can be replicated. It enables load balancing and distribution of services. As FEG is used as a *white box,* the properties of its input language *L* and its output language L_1 must be included in the requirements specifications. It is similar to the specification and design of user interface components (portals), compare Král and Žemlička (2003), and to the generation of temporary services controlling individual business processes. The proper specification of gates and front-end gates substantially simplifies the system specification, documentation, development, and maintenance.

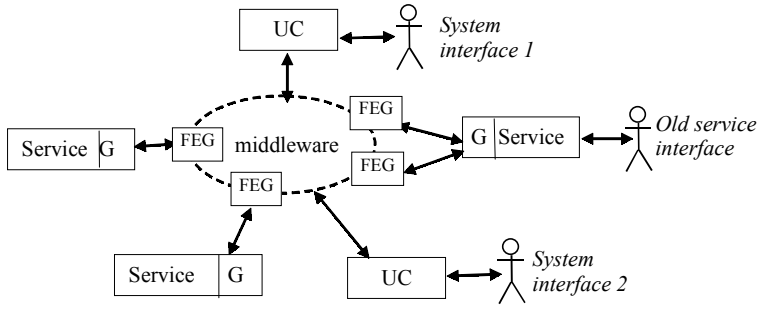
FEG can be viewed as an enhancement of the middleware services as the developers develop, in this case, rather the middleware than the applications (Figure 5, see Král, 1999).

The interfaces of software services should mirror the interfaces of real-life services if possible. P2p systems enable the incremental development strategy starting from the most useful services. The relation of the services to their real-life counterparts enables a reliable estimation of what services are the most useful ones. According to Pareto 80-20 rule, it enables the achievement of 80% usefulness of the system consuming 20% of effort only.

Petri Nets

FEG are based on specific tools and methods having common features with compiler construction and formal language translation. Methodologically, they are similar to user interface services (portals). They can also play the role of message routers. The time of execution of FEG is negligible as there is no waiting on answers from users and/or technologies. FEG can compose several messages into one message and decompose one message into several messages. FEG is a solution of the interoperability problem mentioned by Schoder and Fischbach (2003). FEG can therefore be viewed as a generalization of places in Petri nets with colored tokens (Petersen, 1997). Tokens are

Figure 5. System with front-end gates



messages, whereas application services behave like processes in temporal colored Petri nets. It is open as to how to generalize Petri nets so that they enable a proper modeling (diagramming inclusive) and simulation of confederations. Petri nets describe, in some sense, atoms of communications in networks of static structure. Workflows must therefore be defined using some other tools. Petri nets are used in manufacturing control systems (Vondrák et al., 2001). It again indicates links between software confederations and manufacturing systems (and, generally, soft real-time systems). Confederations are necessary but there is not enough experience with them among computer professionals at enterprise level.

Requirement Specification Issues

The requirement specifications for confederations must often be oriented toward the use of existing systems and their interfaces. The specifications should take into account the properties of the interfaces and their dynamics. It appears that the people with the experience with low-level process control systems could today help a lot at the top-most tier of enterprise systems now having confederative architecture, as the confederative orientation is common for the soft real-time system developers. They should therefore take part in requirements specification of large software systems.

The requirements can now include the new system function types formulated by CEOs like enterprise decentralization, boundaries of divisions, various forms of in- and outsourcing, CRM, purchase coalitions, and so forth. Many such requirements are feasible only if the enterprise information system is confederated and has an appropriate structure. The top management should be aware about it, and the developers should know something about management and its needs. So, there should be no high wall between developers and users (Král & Žemlička, 2003). It contradicts the recommendations from "Recipe for Success" by Standish Group, available at *www.standishgroup.com*. Note that agile programming proposes permanent contacts between developers and users. Software confederations offer the opportunity to use agile programming in the development of large systems provided they are service-oriented.

Confederations are challenge and issues for CIO like the balance between centralized and decentralized agreement of message formats. The concept of software confederations and e-commerce is a large challenge for software vendors and system integrators. They have to change their business strategies.

Other Issues

Software confederations can solve the problem of Reorg Cycle (Armour, 2003) saying that the enterprises are permanently reorganized. As during a lengthy reorganization, the conditions on the market change and a new reorganization is necessary. SO is a solution, as it simplifies and shortens the reorganization.

The first SOSS have long ago manifested their advantages like easy prototyping, incremental development, stability, and flexibility. The mainstream of software engineering has then not been faced with the necessity to build confederations at enterprise level. There was no appropriate technology and no strong need to leave the design viewing the system as one possibly distributed logical unit containing no large black boxes.

The situation has changed due to the progress in hardware and software (Internet). SOSS became technically feasible. At the same time, globalization has generated the need for confederated global enterprises and, therefore, for SOSS.

The services can be cloned and made movable like software agents. It can simplify the design of mobile systems. There are issues common with grid systems.

Paradigm Shift

The construction of any system as a collection of services mirroring the structure of real-world services has substantial advantages:

- As the interfaces are user-oriented, the system specification is simpler due to easier involvement of users and a better structure of the specification. It simplifies the use of the system by users, as they understand what the system offers. The users can then easily modify business processes.
- It supports preferable software engineering properties like openness, maintainability, modifiability, and so forth.
- The services can be specified by their interfaces. It opens the opportunity to apply agile programming (Beck et al., 2001) or extreme programming (Beck, 1999) in large projects.

The decomposition of the activities into autonomous services is a very important invention. It is likely that the use of human-like behaving services in service-oriented systems brings the flexibility and power known from human society.

The main barrier of a wider application of service orientation in our sense is that SO is a new paradigm for the majority of software developers. The acceptance and governance of SO will therefore be a long-term process (remember the case of object orientation). There is no general agreement, even in the definition of the content of service orientation (Barry and Associates, 2003). Many antipatterns from Brown et al. (1998), like "Islands of Automation," "Function Decomposition," and so forth, need not be antipatterns in SOSS anymore. Other antipatterns like "Lava Flow" are not dangerous.

The main attributes of good SOSSs are: (i) system is designed as a peer-to-peer network of services; (ii) the peers mirror real-life services and have user-oriented interfaces; and (iii) it is preferable to have user performable service interfaces.

Information Systems of Manufacturing and Information Systems of Global Enterprises

The automated manufacturing systems have the main features of software confederations; they are (partly) service-oriented. SO has therefore been applied in manufacturing systems for many years. Object orientation (OO) is common now at the enterprise level. OO CASE systems based on UML (OMG, 2001b) and model-driven architecture (MDA) (OMG, 2001a) are used there. SO is common for the lowest enterprise management (manufacturing) systems and should be common on the top management level (management of international enterprises), as well. The middle management (local factories) usually rely on OO methodology (compare Table 1).

The format mismatch of enterprise application interface (EAI) can be resolved by front-end gates. Note that EAI is not primarily intended to support B2B (compare Pinkston, 2002). The fact that manufacturing systems are service-oriented has important consequences. The middle management, usually managing the local units of international enterprises, should not insist on the use of OO methods as a *golden hammer* applicable everywhere as it can lead to the application of OO philosophy outside its applicability. The shift to SOSS on the upper enterprise level is, however, not easy as there is lack of SO experts. It is not optimal to involve here only the people from middle management level. They are not service-oriented. It is difficult for them to *convert* their object-oriented thinking into the service-oriented one. The obstacle is their mental barrier. The nice OO design patterns (Gamma et al., 1993) are, to a high degree, useless in SOSS; OO people often have the feeling that the confederative philosophy is a step back. Such a barrier is exceptional for people having experience with systems including technological process control. The difficulty of the acceptance of SO thinking could be for OO people even more difficult than the conversion from structural thinking to OO thinking (compare Nelson, Armstrong & Ghods, 2002) several decades ago.

Conclusion

Important functions, especially the functions supporting CEO activities, depend on application of service orientation. It implies that the system must have a proper

Table 1. Application domains of object and confederative orientations

Software Technique	Area of Application
Service orientation (possibly partly)	*Manufacturing control level*: CIM components, real-time systems
Object orientation (for example, UML, MDA) still suffices	*Monolithic enterprise level*: middle management, divisions of an international enterprise, highly centralized organizations
Service orientation desirable, necessary, EAI	*Global (world-wide) enterprise level*: international enterprises, state administration ...
SO philosophy necessary, EAI, B2B	*World-wide business*: some health network services, coalition of car vendors, e-business

architecture as a p2p network of autonomous services having user-oriented interface, system user interface services, and eventually, some newly developed infrastructure services (FEG). The SO philosophy therefore influences requirements specification more substantially than other philosophies. This issue is not understood enough. Architectures are very difficult to change. They must therefore be chosen early during the requirements specifications. Vice versa, the requirements must reflect the properties of the architecture.

User-oriented interfaces of services simplify the collaboration between users and developers, enable the development, exploiting the possibility that services can be simulated via user interface services (portals). It simplifies the design of business processes and enhances the software engineering properties of the system. Good interfaces can easily be designed if the services mirror the real-life services.

The user-oriented interfaces can serve as a specification of the services. It is achievable only if the collaboration between users and developers should be tight, and the developers should be able to understand user knowledge domains. It must be trained. The user-oriented interfaces enable the use of different implementation technologies in different services and the agile forms of development in large projects.

Properly chosen system architecture influences and often determines the tools and processes of the system development. An issue is that there are no good modeling tools for SOSS. The main obstacle here, however, is the inability or unwillingness of many developers to apply SO. It is the consequence of the fact that it is as any paradigm shift: a long-term process. Solution can be in the engagement of people already having the service-oriented feeling, for example, of the developers of soft real-time systems.

SO changes the tasks of IT management that should facilitate the agreements of the details of system architecture. IT management can be less dependent on software vendors as it now has a greater freedom of what to buy from whom and what to develop to achieve a competitive advantage. Good software engineering properties of SOSS simplify many tasks of IT management (selective outsourcing, development process control, modifications and maintenance, and so forth). SO requires changes of marketing strategies and methods of software vendors and system integrators.

SOSS developers must often apply data- and object-oriented techniques and even integrate batch applications. The developers must be able to understand and use the

knowledge of users from all the levels of organization hierarchy. It is not easy, and it should be taken into account in the education of software experts, usually too proud of their narrow and detailed computer-oriented knowledge.

SO is a philosophy influencing the whole software industry and practice. It promises to open the way to the software of the quality known from the other branches of industry (no "Warranty Disclaimer").

Acknowledgments

This work has been supported by Czech Science Foundation by grants No. 201/02/1456 and 201/04/1102.

References

Armour, P. (2003). The Reorg Cycle. *Communications of the ACM, 46,* 19-22.

Barry and Associates. (2003). Retrieved August 15, 2004, from *http://www.service-architecture.com*

Beck, K. (1999). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Agile programming manifesto.* Retrieved August 15, 2004, from *http://www.agilemanifesto.org/*

Bray, I. K. (2002). *An introduction to requirements engineering.* Harlow, UK: Addison-Wesley.

Brown, W. J., Malveau, R. C., McCormick, I. H. W., & Mowbray, T. J. (1998). *AntiPatterns: Refactoring software, architectures, and projects in crisis*. New York: John Wiley & Sons.

Datz, T. (2004, January 15). What you need to know about service-oriented architecture. *CIO Magazine.* Retrieved August 15, 2004, from *http://64.28.79.79/archive/011504/soa.html*

Demetriades, J. T. (2003). Does IT still matter. *Business Integration Journal,* 20-23.

Donnay Software Designs (1999). *Mature, portable, data-driven systems.* Retrieved August 15, 2004, from *http://www.dclip.com/datadr.htm*

Dyché, J. (2002). *The CRM handbook: A business guide to customer relationship management.* Boston: Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). *Design patterns. Elements of reusable object-oriented software.* Boston: Addison-Wesley.

Král, J. (1995). *Experience with the development and use of flexible manufacturing systems.* Unpublished manuscript.

Král, J. (1999). Middleware orientation: Inverse software development strategy. In W. Wojtkovski, W. G. Wojtkowski, S. Wrycza, & J. Župančič (Eds.), *Systems development methods for databases, enterprise modeling, and workflow management* (pp. 385-396). New York: Kluwer Academic/Plenum.

Král, J., & Žemlička, M. (2000). Autonomous components. In V. Hlaváč, K. G. Jeffery, & J. Wiedermann (Eds.), *SOFSEM 2000: Theory and practice of informatics*, *Vol. 1963 LNCS* (pp. 375-383). Berlin: Springer-Verlag.

Král, J., & Žemlička, M. (2001). Electronic government and software confederations. In A. M. Tjoa, & R. R. Wagner (Eds.), *Twelfth International Workshop on Database and Experts System Application* (pp. 125-130). Los Alamitos, CA: IEEE Computer Society.

Král, J., & Žemlička, M. (2002). Component types in software confederations. In M. H. Hamza, (Ed.), *Applied informatics* (pp. 125-130). Anaheim: ACTA Press.

Král, J., & Žemlička, M. (2003). Software confederations - An architecture for global systems and global management. In S. Kamel, (Ed.), *Managing globally with information technology* (pp. 57-81). Hershey, PA: Idea Group.

Lenzerini, M. (2001). Data integration is harder than you thought. Retrieved August 15, 2004, from *www.science.unitn.it/coopis, choice videos/slides*

Lowson, B., King, R., & Hunter, A. (1999). *Quick response: Managing the supply chain to meet consumer demand*. New York: John Wiley & Sons.

Nelson, J., Armstrong, D. A., & Ghods, M. (2002). Old dogs and new tricks. *Communications of the ACM*, *45*(10), 132-136.

OMG. (2001a). Model driven architecture. Retrieved August 15, 2004, from *http://www.omg.org/mda*

OMG. (2001b). Unified Modeling Language. Retrieved August 15, 2004, from *www.omg.org/technology /documents/formal/uml.htm*

Peterson, J. L. (1997). Petri nets. *ACM Computing Surveys*, *9*(3), 223-251.

Pinkston, J. (2002). The ins and outs of integration, how EAI differs from B2B integration. *e-I Journal*, 48-52.

Rowe, D. (2002). E-government motives and organizational framework. In J. Pour, & J. Voříšek (Eds.), *Systems integration 2002, Conference presentations* (pp. 93-99). Prague University of Economics, Prague, Czech Republic.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design*. Englewood Cliffs, NJ: Prentice Hall.

Schoder, D., & Fischbach, K. (2003). Peer-to-peer prospects. *Communications of the ACM*, *46*, 27-29.

Vondrák, I., Kružel, M., Matoušek, P., Szturc, R., & Beneš, M. (2001). From business process modeling to workflow management. In M. Bieliková (Ed.), *DATAKON 2001* (pp. 241-248). Brno.

W3C. (2001). Web service definition language. A proposal of W3 Consortium. Retrieved August 15, 2004, from *http://www.w3.org/TR/wsdl*

W3C. (2002). Web services activity. Retrieved August 15, 2004, from *http://www.w3.org/2002/ws/*

Yourdon, E. (1988). *Modern structured analysis* (2nd ed.). Prentice Hall.

Endnotes

[1] Other solutions are not feasible for technical as well as for practical reasons (Král & Žemlička, 2001, 2003; Rowe, 2002).

Chapter X

Service Patterns for Enterprise Information Systems

Constantinos Constantinides
Concordia University, Canada

George Roussos
University of London, UK

Abstract

This chapter introduces service patterns for SOA-based enterprise systems. The authors believe that the deployment of such patterns would be of considerable value both as a best-practice guide for practitioners as well as a starting point for further research in their role in software engineering. A comprehensive catalog of service patterns is included in this chapter. In the catalog, each pattern is discussed in the context of selected examples and in terms of a brief description of its role, functionality, and deployment. For each pattern there are recommendations on implementation and a practical usage scenario.

Introduction

Modern enterprise information systems constitute a core component in business support. Not only must they provide reliable infrastructures for the organization itself but they also must be capable of seamlessly connecting to the systems of other businesses in their value added network of partnerships. These operating conditions

demand that enterprise systems should operate transparently and should be flexible. Over the past few years, service-oriented architectures (SOA) have emerged as a framework that addresses this requirement both effectively and efficiently. Furthermore, time after time systems developers discover that successful and cost-effective design of information architectures requires a combination of theory and practical experience as well as reuse of robust and proven designs that form solutions to frequently occurring problems. Effective reuse can speed up the development process, reduce costs, increase productivity, and improve the quality of software.

Design-Level Reuse

As software products need to satisfy both technical and nontechnical criteria, developers find it essential to combine theory and experience in order to reuse proven designs. The importance of reuse lies on the fact that it can speed up the development process, cut down costs, increase productivity, and improve the quality of software. Design-level reuse is viewed as the attempt to share certain aspects of an approach across various projects. There is, however, no single approach that can support reuse across all architectural layers. Object- and component-based systems offer a wide spectrum of techniques to reuse designs on different levels, ranging from system architectures, frameworks, and patterns to libraries and programming languages. Developers can view reuse on a range of different levels of granularity, ranging from effective consistency sharing in low-level programming to sharing subsystem architectures or overall structure. Libraries and languages constitute an approach on how to best program *in the small.* Patterns, on the other hand, fall in the part of the spectrum that supports the sharing of interaction architectures, and they constitute an approach on how to best program *in the large* (Szyperski, 2002).

Patterns

The notion of a pattern originates from the discipline of Architecture from the book by Alexander et al. (1977), and it refers to solutions of occurring problems in the constructions of towns and buildings. Along the same lines, a pattern in Software Engineering refers to a solution to a recurring problem that arises during software development that can be used in different contexts. This definition leaves space for overloading and the term, indeed, tends to be overloaded in the literature, as patterns are becoming available over a wide spectrum of granularity, ranging from very general design principles to language-specific idioms.

- ***Design patterns:*** Design patterns are micro-architectures that describe the abstract interaction between objects collaborating to solve a particular problem. They have become popular through the seminal book by Gamma et al. (1995), also referred to as the Gang of Four (GoF). In Gamma et al. (1995), each pattern is discussed in terms of purpose (what the pattern does) and scope (whether the pattern applies primarily

to classes or objects). Furthermore, patterns are categorized into creational, structural, and behavioral.

- ***Responsibility assignment patterns:*** GRASP (General Responsibility Assignment Software Patterns) proposed by Larman (2002) are a set of design principles on the assignment of responsibilities to objects where a responsibility refers to the obligation of an object in terms of its behavior. GRASP patterns are centered on the notion of two types of responsibilities: doing and knowing.
- ***Service patterns:*** With the wide availability of enterprise information systems and Web services, service patterns (Monday, 2003) are introduced to identify abstract problem-solving scenarios that involve Web services as the appropriate solution, particularly in view of their property to expose state management services in the context of a heterogeneous network of client components. The effectiveness of service patterns lies in their ability to address a design problem in many different domains, thus providing a generic solution that may be available and can be adapted in various contexts. Unlike design patterns which define micro-architectures by describing the abstract interaction between objects that operate collectively to solve one particular problem, service patterns encompass larger segments of functionality in a decentralized and heterogeneous context. Moreover, service patterns can effectively be used as a vocabulary between developers who can use combinations of patterns to develop service pattern languages in a manner similar to design pattern languages.

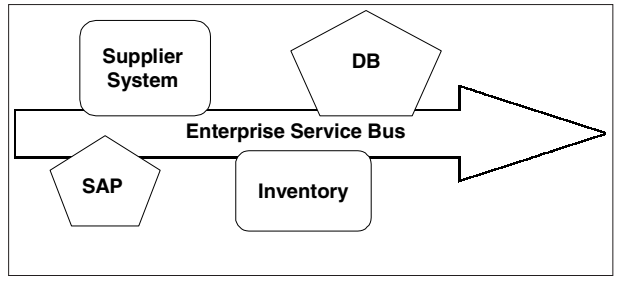
Overall, patterns can help developers reuse successful designs based on proven experience, as well as provide good documentation and a high level of maintenance of systems.

Background

The Service-Oriented Architecture itself is often seen as a design pattern. Its value lies in the fact that it can abstract enterprise architectures using the concept of the so-called Enterprise Service Bus (ESB), a shared communications medium on which services may connect to and use in a plug-and-play manner (Figure 1). This may be thought of as the equivalent of a bus in a computer architecture, which provides the foundation for core and peripheral components to connect to and communicate transparently with each other. Different internal and external systems may connect to the bus transparently.

Enterprise computing is one of the most complex and challenging computing and communications environments in use today. Indeed, enterprise systems are highly decentralized and heterogeneous, and they are frequently geographically distributed with global scope: (1) they need to be scalable to cater to large numbers of changing users; (2) they have to be highly reliable to effectively support business processes (indeed, systems failure has distinct and measurable business impact); (3) they have to be trustworthy so as to protect confidential information on which business operations

Figure 1. Enterprise service bus



depend; and (4) they have to be heterogeneous in terms of technologies and systems at all levels from application all the way to hardware. In particular, heterogeneity and distribution imply that there are frequent nontrivial issues on synchronization and concurrency as well as compatibility.

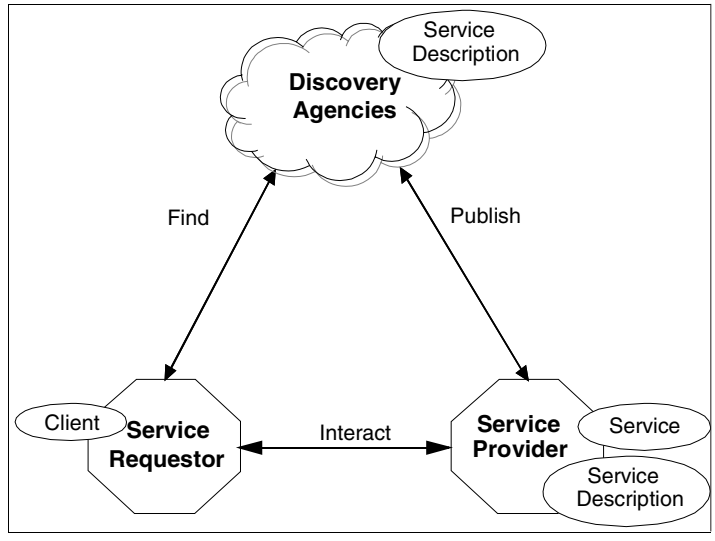
To address these issues, numerous frameworks have been developed over the past decades, which employ middleware services and may rely heavily on reusable code and design patterns. Such frameworks need to address multiple issues including efficient and effective handling of remote processes, data and input/output; naming; brokering, trading, and leasing resources; multiple levels of software abstractions; multiple attributes; security and trust management; threading and synchronization; and finally, distributed transaction processing. In this context, SOA provides a novel solution which offers a significant advantage over all other solutions, namely, its conceptual simplicity. Indeed, the level of transparency offered by the SOA pattern is unprecedented in enterprise systems.

Further to the conceptual model offered by the SOA pattern for systems architectures, a question naturally arises on the actual implications of the model for efficient implementation. This architecture is often represented as a triangle (Figure 2) with the three main participating actors at the apexes: the service provider, the service requestor, and the discovery agencies (Kreger, 2003). The latter is responsible for providing the illusion of plug-and-play operation: it responds to requests for service, matches up the request with a suitable provider, and provides the rules that define the interaction between them.

The SOA pattern provides a mediation mechanism which abstracts systemic behavior, thus removing the need for engineering and coordinating resources which is a common feature of framework-based approaches. It is therefore a natural consequence that most of the service patterns discussed in this chapter provide mediation mechanisms for different contexts and uses. Service-Oriented Architectures bring under a common umbrella the need for access, location, concurrency, replication, failure, migration, performance, and scaling transparency.

All service patterns discussed in this chapter are intended to provide solutions for common cases for the support exactly this type of transparent operation in different circumstances. Our primary aim is to show that service patterns provide a useful abstraction for the development of such architectures, and they are therefore a useful tool for both researchers and practitioners.

Figure 2. Service-oriented architectures



Furthermore, we aim to discuss a selection of service patterns following a structured and consistent format, choosing from patterns that have emerged in the last few years and have proven to be effective in a variety of circumstances. In doing this, we expect to develop a catalog of service patterns which can be used as a reference for systems developers who wish to examine and possibly implement particular patterns as solutions to practical problems. Moreover, we anticipate that this service pattern catalog will be a useful starting point for software engineering researchers who wish to further develop and document this emerging area. Finally, we will discuss specific case studies drawn from both industrial and research projects where each of the presented service patterns has proven useful.

In the remaining sections of this chapter, we will present a selection of service patterns. We have classified these patterns as structural, behavioral, and concurrency. For each pattern the presentation will include a brief description of its role and an illustration of their functionality with the aid of UML diagrams. We also provide recommendations on their deployment and implementation and discuss a practical usage scenario for each.

A Service Pattern Catalog

We have selected eight structural, behavioral, and concurrency patterns for inclusion to this catalog. To differentiate between the different types of service patterns, it is useful to refer to the Web Services Six-Layer Architecture (Table 1).

Structural Service Patterns refer to the composition of services into larger structures, potentially also constructed as Web services themselves:

- The **proxy** service pattern forces messages to a service to be delivered indirectly through a surrogate service which remains transparent to clients of the service. The proxy pattern operates at the messaging layer, and it is oblivious of the semantics of the involved Web services.
- The **façade** service pattern simplifies access to a related set of Web services by providing one contact point that all clients use in order to communicate with the set, effectively reducing the complexity of interactions with it.

Behavioral Service Patterns define different modes of interaction as well as relationships between Web services. Unlike structural patterns, which operate at the messaging layer, behavioral service patterns operate at the workflow, discovery, or registry levels.

- The **infomediator** service pattern uses one public service point to coordinate state changes between a set of services and at the same time to maintain low coupling between them. The infomediator pattern operates at the Registry layer and provides translation and coordination functionality using application semantic knowledge (for example, product catalog data) (Roussos, 2004).
- The **observer** service pattern provides a one-to-many dependency among Web services so that when the subject service changes state, its dependents (observers) are notified and maintain a state in synchronicity with the subject. In this case, state refers to workflow, discovery, and registry layer status.
- The **strategy** service pattern encapsulates related behavior in Web services that are derivatives of a common service template. Thus, a request for a higher-layer operation may be translated in different fulfillment strategies according to the specific context, transparently for the client.
- The **marketplace** service pattern interfaces transparently with both service consumers and providers and implements pair-wise matching algorithms (for example, auctions or reverse auctions) to balance service requirements and offerings.

Concurrency Service Patterns aim to organize exploitable concurrency between Web services.

Table 1. Web service six-layer architecture (adapted from WebServices.Org)

LAYER	PROTOCOL
Service Negotiation	Trading Partner Agreement (legal document)
Workflow, Discovery and Registries	UDDI, ebXML registries, IBM WSFL, MS XLANG
Service Description	WSDL, WSCL
Messaging	SOAP, XML RPC
Transport	HTTP, HTTPS, FTP, SMTP
Network	TCP/IP, Diffserv, RSVP, SSL/TLS, SNMP

- The **producer-consumer** service pattern observes the concurrency protocol with the same name and coordinates the asynchronous production and consumption of information resources.
- The **retrieve-update lock** pattern, based on Lea (1999), observes the readers-writers protocol which allows concurrent access to a particular service for retrieval while it requires exclusive access for update operations. Variations of the protocol refer to the allocation of priorities (for example, updates may have priority over retrieval).

In the following sections, we will discuss each of the eight service patterns in turn, in terms of the following views: (1) **intent**: what is the intended function of the pattern; (2) **forces**: what is the motivation behind the development of this pattern, and it would usually include a real work example; (3) **applicability**: in which situations is it appropriate to use this particular pattern; (4) **structure**: what are the different elements of the pattern, and how do they interact with each other; (5) **collaborations**: how may this pattern be used with others to construct more complex solutions; and (6) **consequences**: what are the architectural, performance, overhead, or other implications of the deployment of the specific pattern.

Proxy Service Pattern

- **Intent:** The proxy pattern is used to provide a surrogate for the actual service provider in a way that its existence is transparent to clients. Even though clients interact with the proxy, they are unaware of its existence.
- **Forces:** Interacting with proprietary systems or frameworks places considerable restrictions on the flexibility of development, especially when one is dealing with legacy systems or deprecated frameworks. The proxy pattern provides a Web service interface to proprietary interfaces behind an appropriately constructed service point. A proxy usually implements an equivalent interface as the proprietary system it serves as a surrogate to. The proxy intercepts incoming messages, translates them to the proprietary interface, and passes them to the actual system, often after possibly performing certain additional tasks
- **Applicability:** The best use of the proxy pattern is in prolonging the usefulness of enterprise legacy systems by enabling their access over the SOA enterprise bus. Rather than re-implementing the subsystem that supports certain (possibly mission critical) business processes, such systems are hidden behind a proxy service, which acts effectively as its connector to the service architecture. Variants of the pattern are protection proxies (controlling access to the original service), or smart references (taking additional actions when the target service provider is accessed). An example is the Beauty Shop cosmetics retail site (URL: http://www.beautyshop.gr), which is supported by legacy IBM mainframes in an enterprise system used for over two decades to provide retail operations for a brick-and-mortar chain of retail shops. The shop catalog data can be made available as a web

service to an extensive network of resellers. Thus, access to catalog data would be made possible via a standard interface and clients would not require direct communication with the back end.

- **Structure:** The interactions of the participants of the proxy service pattern are shown on the UML sequence diagram in Figure 3(a). A UML class diagram that illustrates the static structure of the pattern is illustrated in Figure 3(b).
- **Collaborations:** The proxy may interact with a service provider service or delegate requests to it.
- **Consequences:** The proxy pattern provides a level of indirection (transparent to clients of the target service) when clients request services from a provider service. The consequences of the level of indirection are based on the type of proxy.

A virtual proxy may implement performance optimizations, a remote proxy may hide the actual location of the service provider and a protection or smart reference proxy may allow additional tasks to be performed when the service provider is accessed. The performance penalty that a proxy service pattern may introduce should be evaluated against the additional development effort and resources required for the re-engineering of the legacy system it abstract.

Figure 3(a). UML sequence diagram for the proxy service pattern

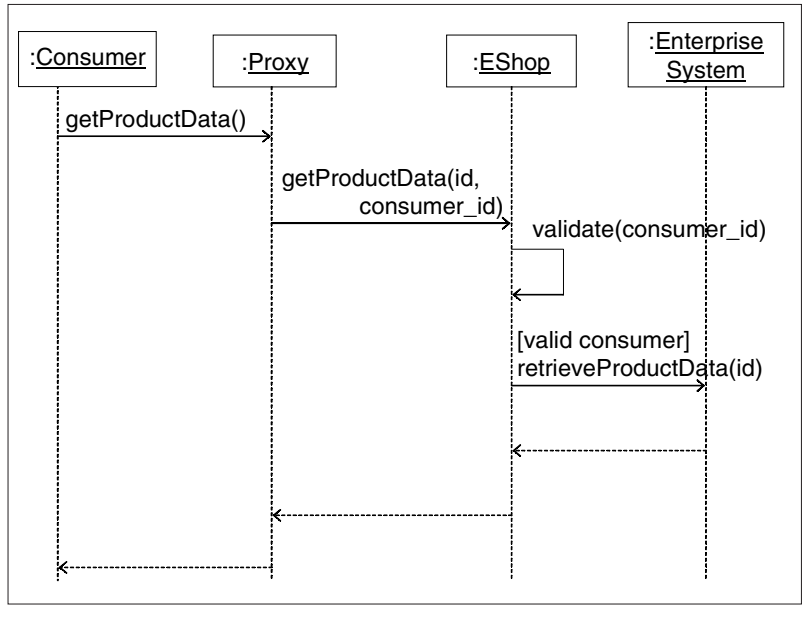
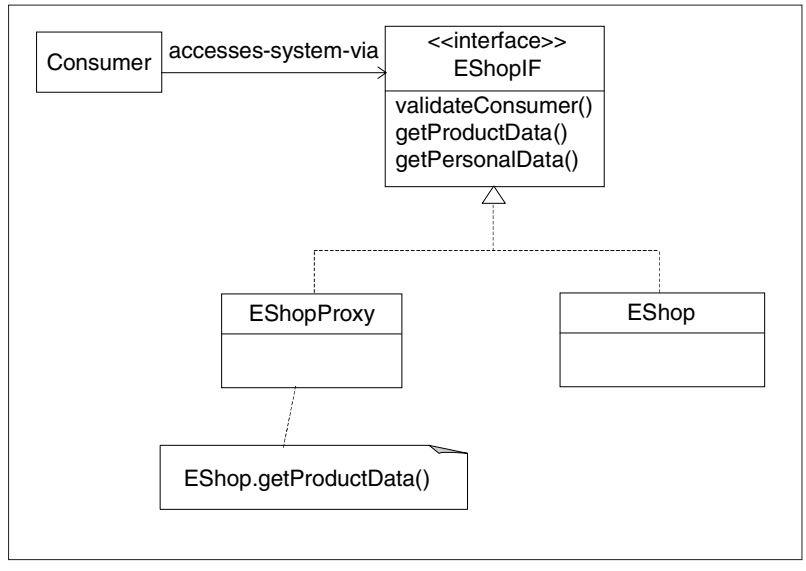


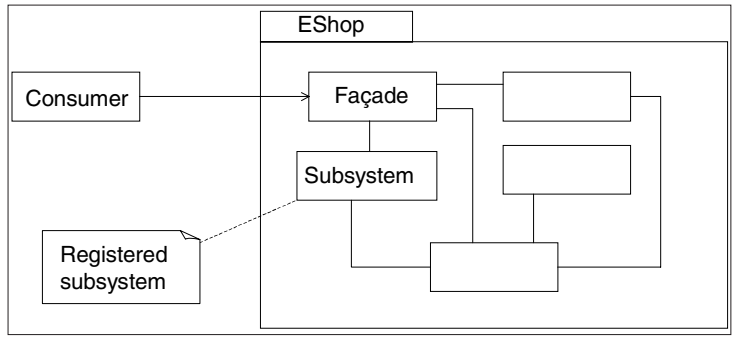
Figure 3 (b). UML class diagram of the proxy service pattern.



Façade Service Pattern

- **Intent:** The façade pattern defines a service point for a subsystem that clients can use in order to communicate with that subsystem.
- **Forces:** Within a subsystem, the dependencies between components add complexity to clients. The façade pattern provides a service endpoint that hides most of the complexity of directly interacting with the individual elements of the subsystem. Clients need only be aware of the interface provided by the façade service.
- **Applicability:** The façade pattern can be used to provide a common service access point to a collection of architecturally different end systems. Clients then can use a single, standard service partner rather than having to implement a variety of different protocols. For example, Amazon (URL: www.amazon.com) brings together under the same roof catalogs of different consumer good categories, for example books, electric goods and travel. The different elements of the catalog are hosted in different subsystems implemented on different platforms using different frameworks (this is a result of the business acquisition strategy of Amazon rather than a sound systems architecture principle). However, the full Amazon catalog is available to third parties as a searchable Web service.
- **Structure:** The UML class diagram that illustrates the static structure of the façade service pattern is illustrated in Figure 4.
- **Consequences:** The deployment of the façade design pattern simplifies access to the service, as clients do not need to know the implementation details of the underlining systems.

Figure 4. Structure of the façade service pattern



Infomediator Service Pattern

- **Intent:** The infomediator pattern defines a service that encapsulates how a collection of services interacts. The infomediator promotes loose coupling and decreased system complexity by keeping objects from referring to each other explicitly, and it allows varying their interaction independently.
- **Force:** Often, complex interaction patterns within a collection of services results in an increased system complexity, as each service requires knowledge of every other service. As a consequence, overall system behavior tends to be distributed among different system components, making it difficult to modify even a single element.

Reusability and adaptability of individual services is low, as even minor modifications require considerable changes to all services interacting with it. The infomediator pattern provides a service endpoint whose responsibility is twofold: (1) to encapsulate the rules of interaction between services and (2) to coordinate the interactions of a collection of communicating services by providing a centralized point of access, thus preventing services from having to explicitly refer to others. Services do not need to know about each other; they just need to know their infomediator and how to interact with it.

- **Structure:** The interactions of the participants of the infomediator service pattern are shown on the UML sequence diagram in Figure 5(a), and a UML class diagram that illustrates the static structure of the pattern is illustrated in Figure 5(b).
- **Consequences:** The clients are the primary benefactors of the infomediator pattern since they may participate in complex interaction scenarios without the overhead of adaptation to each peer separately. The infomediator service itself may implement different interaction strategies or policies, thus modifying the behavior of the overall system transparently for the clients.

Observer Service Pattern

- **Intent:** The observer pattern maintains a one-to-many dependency between services. This service pattern notifies dependent services, the observers, for changes in the state of the observed service, the subject, without exposing the state of the subject itself.
- **Forces:** A subject may have any number of dependant observers. Once a subject generates an event marking a change of state, all observers receive notifications and, upon checking appropriate conditions, proceed to take a predefined action. Often, conditions relate to composite events, that is, they depend on two or more events joined by logical operators.

This pattern should be used when a change in state on one service requires a change in an unanticipated number of others. In more complex interaction situations, subjects may be joined in collections and observers may be notified of events of common types generated by any one of the subjects. In this case, we have the publish-subscribe observer service pattern.

- **Applicability:** Workflow applications may often benefit by the use of the observer pattern. In particular, when such applications are built on top of XML documents, the observer service pattern may be used to synchronize them. For example, when

Figure 5(a). UML sequence diagram of the infomediator service pattern

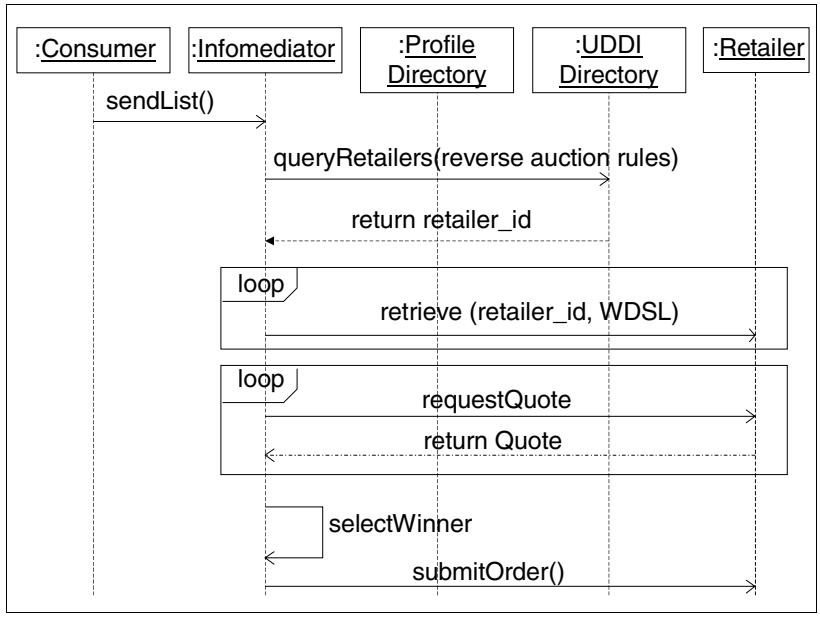
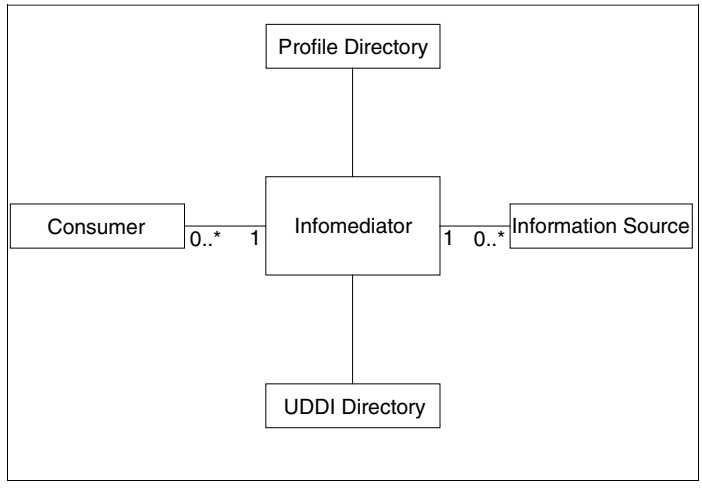


Figure 5(b). UML class diagram of the infomediator service pattern



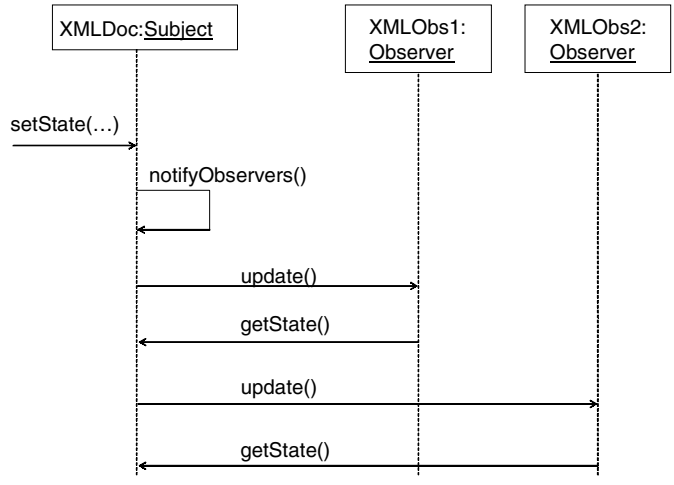
business process descriptions are maintained as distributed XML representations and changes to a particular process element are applied, this service pattern would generate events which will trigger a chain of changes throughout the descriptors according to the defined rules. The use of this pattern has proven particularly useful in the context of Event-Condition-Action (ECA) rules for the synchronization of learning objects formatted as XML metadata (Papamarkos et al., 2003).

- **Structure:** The interactions of the participants of the observer service pattern are shown on the UML sequence diagram in Figure 6(a), and a UML class diagram that illustrates the static structure of the pattern is illustrated in Figure 6(b).
- **Consequences:** There is a low coupling between subject and observer services. Furthermore, observers may be functionally different and unrelated services as long as they follow the pattern requirements. The responsibility of how to handle a notification relies completely with the observer. The observer service pattern implementation also allows for broadcasting. An undesirable result in the deployment of this pattern is that a delivery notification may take a relatively long time

Strategy Service Pattern

- **Intent:** The strategy pattern supports the definition of a set of related algorithms, their separate encapsulation in independent services, and their interchangeable use by allowing them to vary independently of their clients.
- **Forces:** The strategy service pattern relates to the case when there are multiple pathways through an enterprise system for the completion of a specific process. Different circumstances require that the process be better serviced by selecting one of these pathways depending on the local context. Strategy removes the

Figure 6(a). UML sequence diagram of the observer service pattern



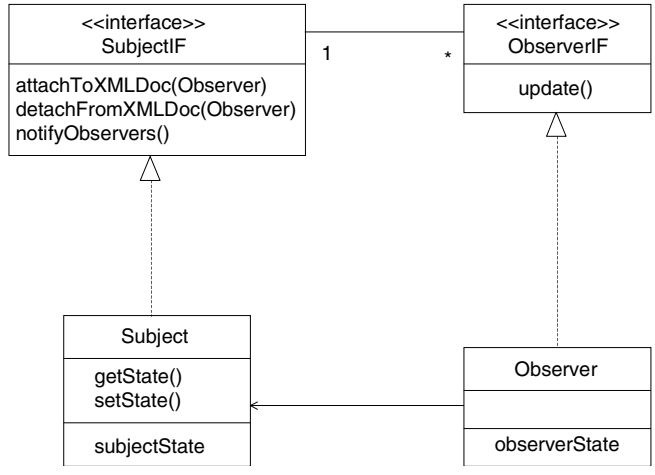
complexity of dealing with the specifics of each service request and taking an appropriate decision by disconnecting the variant services from the enterprise bus and controlling access via a single path.

- **Applicability:** Searching corporate resources for specific information may provide the best illustration of the strategy pattern. Different searching approaches are more appropriate depending on the type of data searched, for example, Page Rank (http://www.google.com) (Brin & Page, 1998) is more appropriate for Intranet Web pages; social network-based approaches (http://www.teoma.com) are more appropriate for e-mail searches; and Navigation Zone (http://www.navigationzone.net) (Wheeldon & Levene, 2003) search is better suited to relational database investigation. A client should not be aware of these different approaches but can only query once for a particular area of interest, and the strategy service pattern will select the most appropriate way to carry out the search.
- **Structure:** The UML class diagram that illustrates the static structure of the strategy service pattern is illustrated in Figure 7.
- **Consequences:** The strategy service pattern supports transparent operation while, at the same time, it offers increased flexibility. Remote state and implementation specifics are abstracted inside the strategy pattern, and thus clients can operate remaining oblivious of such details.

Marketplace Service Pattern

- **Intent:** The marketplace service pattern interfaces transparently with both service consumers and providers and implements pair-wise matching algorithms, aiming to optimize patterns of demand and supply for a specific service.

Figure 6(b). UML class diagram of the observer service pattern



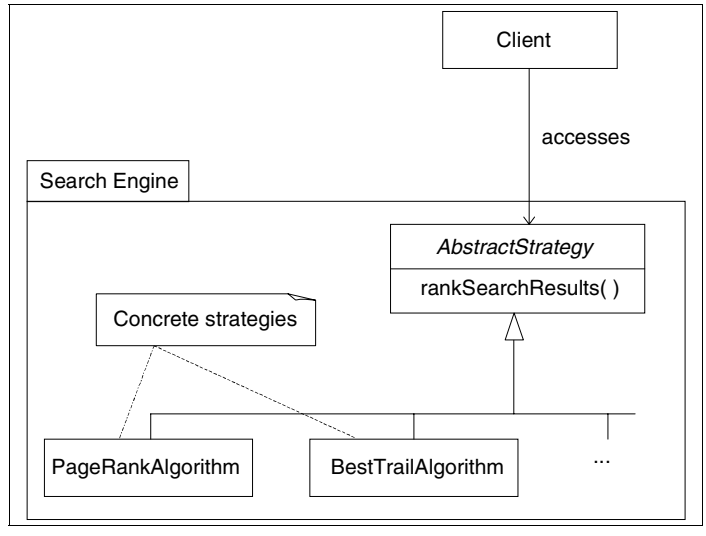
- **Forces:** Two actors interact with the marketplace: service consumers and service suppliers. The role of the consumer is to register and advertise a list of requirements for which suppliers bid competitively according to rules defined by marketplace policies. Both consumers and suppliers interact directly only with the marketplace service and may remain anonymous to each other until the end of the bidding process.

Consumers interact with the marketplace in order to: (1) register (and authenticate); (2) initiate a request for a matching supplier; and (3) receive notification on the end of the competitive bidding process and the name of the winner. The supplier interacts with the service in order to: (1) register its interest (and authenticate); (2) receive notification that possible matches are available; (3) bid for suitable matches and receive notifications of success or failure (this step may be repeated several times); and (4) receive notification on the end of the matching process and the final result.

The marketplace implements access control and directory functionality, a common vocabulary for resource descriptions, WSDL interfaces for both consumer and supplier interaction, and last but not least, application-specific business logic that defines the rules of matching suppliers to consumers. The latter is transparent to both consumers and suppliers and may change according to the context of the transactions performed.

- **Applicability:** The marketplace pattern is useful when many-to-many relationships are involved, for example, in reverse auctions for hospitality services procurement. To support their operations, hotels require a constant stream of products, frequently supplied by different vendors or vendor consortia. Procurement is thus a

Figure 7. UML class diagram of the strategy service pattern.



core component of their operation and, indeed, a process that requires significant efficiency optimizations. For this reason, in the past few years several business-to-business exchanges have been established to provide such procurement services (For example, http://yassas.com). Such exchanges match consumer needs to supplier offers via a competitive bidding process, but because they are implemented on proprietary platforms, they provide either a Web-based interface which requires significant manual intervention by the user or require that the participating actors modify their systems to implement vendor-specific interfaces (Nartovich & Cunico, 2002). Due to the limitations of the Web-based approach and the increased costs of the proprietary systems, both suppliers and consumers increasingly demand that exchanges integrate their systems by providing Web service interfaces. In this context, the marketplace pattern can be used to provide a suitable paradigm for the architecture of such exchanges.

- **Structure:** The interaction of the participants involved in the marketplace service pattern is illustrated in Figure 8.
- **Collaborations:** The robustness factor of the marketplace service pattern implementation may be greatly improved by the combined use of two mechanisms. First, marketplaces may protect their internal state from possible errors caused by unpredictable interruptions of network service by including a client-generated, globally-unique identifier (GUID) that can be stored and checked against at a later time. The GUID provides for the identification of duplicate messages that must be discarded so as to preserve the consistency of the marketplace internal state. Second, the marketplace should implement an event-based queuing mechanism that processes incoming requests so as to provide for high concurrency, robustness, and predictable behavior. Thus, the marketplace service pattern is frequently implemented in conjunction with the ESB pattern.

- **Consequences:** Clients are separated in distinct classes according to their role. Each class needs only be aware of the rules of interaction with the marketplace and changes in these rules affect only this class locally. The marketplace service may modify its internal strategies for matching requests between different classes as appropriate, transparently to all clients. The usefulness of the pattern increases with the number of clients since it reduces the complexity of pair-wise interactions from order N^2 to order N, for N clients. The performance overhead due to indirection should always be offset against this reduction in complexity.

Producer-Consumer Service Pattern

- **Intent:** The producer-consumer service pattern coordinates the asynchronous production and consumption of items placed in a (bounded) information repository, which acts as a shared resource. A producer client may only place items in a nonfull repository. Similarly, a consumer client may only retrieve items from a nonempty repository.
- **Forces:** The producer-consumer service pattern provides an effective and efficient mechanism for decoupling system elements, thus removing the need for them to execute in lockstep. It can be used to provide interoperability between messaging systems, especially in the context of a heterogeneous environment. For example, using the producer-consumer pattern, it is possible to provide flexibility in implementing event persistence as well as to support guaranteed message delivery and queue management. Moreover, this service pattern can enforce event ordering and discard policies as well as provide limitations of message lifeline.
- **Applicability:** Often, transactions that execute in a distributed or heterogeneous environment may have components that require significantly longer completion times. Effectively, that singular transaction element will define the overall time for transaction execution and may prolong its lifetime beyond some acceptable limit. For example, every business-to-consumer (B2C) e-commerce site supports a consumer shopping cart, the data of which at checkout must be used to update inventories to be forwarded for fulfillment and payment received as part of the contract conclusion between the shop and the customer. However, the use of electronic payment methods, for example, Barclay's e-PDQ, or other third-party payment service providers may take up to several minutes to complete, time which is considerably longer than the expected time (normally a few seconds) for user interaction. In this case, the producer-consumer service pattern may be used to queue payment instructions for a later time and proceed with the user interaction scenario. This behavior may be made part of the contract with the consumer as it frequently is, for example, http://currys.com, the digital branch of a UK high street retailer of electric goods, defines in its terms and conditions that submitting an order is treated as an offer to buy on the side of the consumer, and the supplier does not accept it until after funds have been transferred to its accounts.
- **Structure:** The interactions of the participants of the producer-consumer service pattern are shown on the UML communication (collaboration) diagram of Figure

Figure 8(a). Initializing a session with the marketplace service pattern

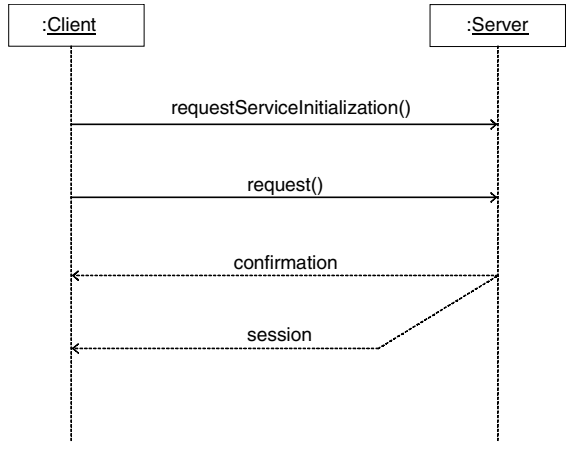
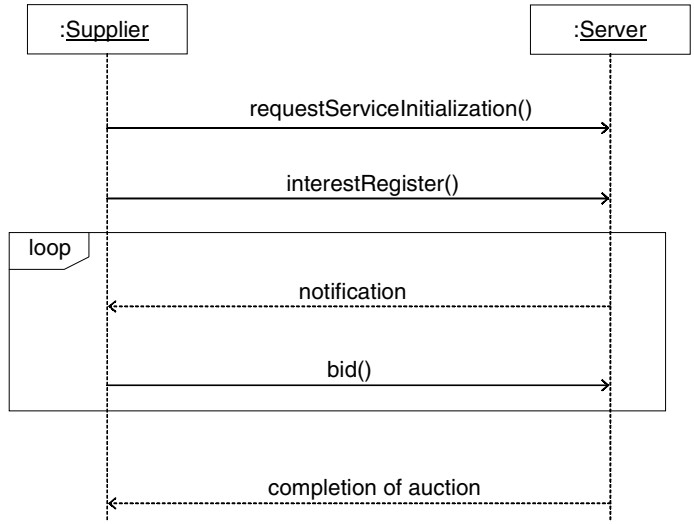


Figure 8(b). UML sequence diagram for the marketplace service pattern



9(a). An alternative design of this pattern can be adopted by taking into consideration that the implementation of the synchronization policy would end up cutting across the various methods of the DataWarehouse class. In general, synchronization is a concern that is inherently crosscutting. There are a number of implications of crosscutting such as: (1) low cohesion of modular units resulting in a low level of comprehensibility of code; (2) strong coupling between modular units

resulting in classes that are difficult to change and where changes in code are difficult to trace; (3) low level of reusability of code; (4) low level of system adaptability; and (5) programs that tend to be more error prone. Aspect-Oriented Programming (AOP), introduced by Kiczales et al. (1997) is a term adopted to describe an increasing number of technologies and approaches that support the explicit capture of crosscutting concerns (or aspects) whereby the implementation of functional components and crosscutting concerns is performed (relatively) separately, and their composition and coordination (referred to as weaving) is specified by a set of rules. Even though AOP is not bound to object systems, most of the current approaches involve extensions to current object-oriented languages that provide linguistic support for the explicit definition of crosscutting concerns, together with special compilers (weavers) that can combine them with components. In this example, we can place the synchronization policy of the producers-consumers protocol in an aspect (and within a separate package) with a unidirectional visibility over the functional component of the system (class DataWarehouse). A UML class diagram that illustrates the aspect-oriented version of the static structure of the pattern is illustrated in Figure 9(b).

- **Collaborations:** The façade service pattern may be used to provide federated event channel and may be used in conjunction with the proxy pattern to connect to legacy endpoints. Finally, the strategy service pattern may provide different implementations, for example, a choice of peer-to-peer or publish-subscribe architectures for the design of producer-consumer systems.
- **Consequences:** Items may be placed in the repository even if no consumer is available at the time. To prevent overflow, production requests first inquire and then possibly wait on condition of a full repository. Similarly, consumption requests should wait on an empty repository. When used in the context of messaging, this service pattern removes the need for solutions based on proprietary application programming interfaces or frameworks and may provide bridging services for CORBA, Java Messaging Service, and other message-oriented middleware solutions.

Retrieve–Update Lock Service Pattern

- **Intent:** The retrieve-update service pattern is based on the readers-writers concurrency protocol, and it enforces safe access to shared resources. To maintain data integrity, a writer client would operate with self and mutual exclusion, whereas readers would operate with mutual exclusion only.
- **Forces:** This service pattern addresses the need to concurrently access a shared resource. The logic for coordinating read (retrieve) and write (update) operations is encapsulated in the service provider and should be reusable, adaptable, and clearly should ensure that starvation of clients is not impossible.
- **Applicability:** The retrieve-update service pattern provides synchronization services for messaging systems. It can therefore provide the foundation for the

Figure 9(a). UML communication diagrams for operations placeItem() and retrieveItem()

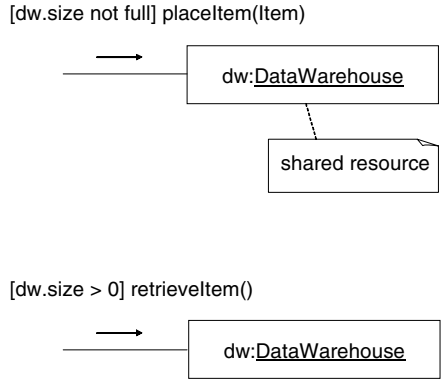
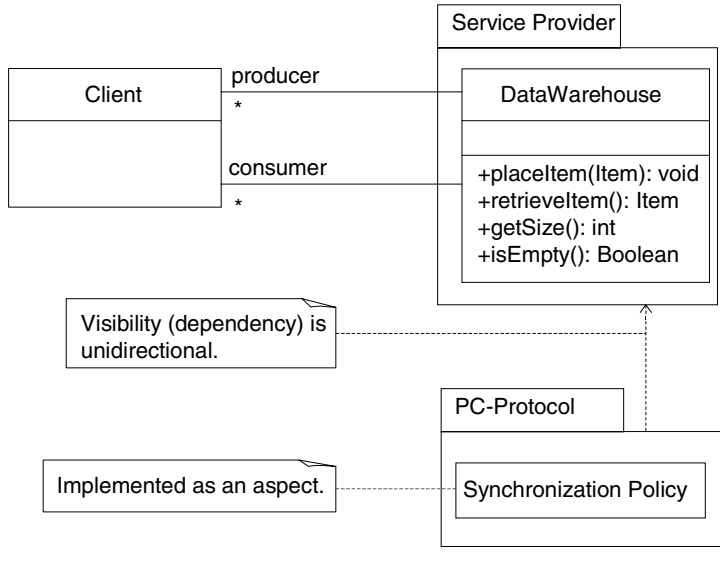


Figure 9(b). UML class diagram for the (aspect-oriented) producer-consumer pattern



construction of the ESB architecture by protecting the integrity of information barriers across systems. Thus, it is often seen as a component of the infrastructure.

Moreover, in the case of an air-travel booking portal like http://www.bookers.com, correctness of prices at the time of purchase can be guaranteed using a retrieve-update

pattern. In case a change of price is initiated by one of the participant airlines, the retrieve-update pattern can guarantee that no sales are possible using the deprecated price, thus, guaranteeing the integrity of all cooperating systems.

- **Structure:** The interactions of the participants of the retrieve-update service pattern are shown on the UML communication diagram in Figure 10(a) for the read (retrieve) operation and in Figure 10(b) for the write (update) operation. This pattern may also adopt an AOP implementation along the lines of the producer consumer pattern, thus, treating synchronization and scheduling as aspects whose implementation will be placed in a set of classes in a separate package.
- **Collaborations:** The retrieve-update service pattern can be deployed to improve robustness for the infomediator and marketplace patterns. For example, in the case of time-sensitive bidding processes, the retrieve-update service pattern offers a mechanism that guarantees protection and linearity of transactions.
- **Consequences:** The retrieve-update service pattern may provide significant efficiency improvements by helping to avoid unnecessary waiting in accessing shared resources. It can also result in significant improvements in two areas: (1) safety of system operation by protecting from interference and (2) improving systems liveliness by preventing starvation of business processes.

Future Trends

In this chapter, we discussed the use of service patterns in service-oriented enterprise systems. We believe that the use of service patterns in this context offers significant advantages and, thus, a catalog of service patterns would be of considerable value both as a best-practice guide for practitioners and also as a starting point for further research in their role in software engineering. To this end, we introduced a brief catalog of some important service patterns to initiate this discussion. Each pattern was discussed in terms of a brief description of its role in the context of selected examples, and its functionality was illustrated in terms of applicable UML diagrams. We also presented recommendations on their deployment and implementation and discussed practical usage scenarios.

This discussion highlights several commonalities between the different patterns. Indeed, in many cases, they have a meditative role at various levels between systems components, and they effectively support transparency in distributed and heterogeneous situations. Hence, different combinations of the service patterns can be used in the context of diverse enterprise environments to tailor the implementation of the ESB to the particular requirements of specific organizations. Even though, conceptually, the ESB abstraction offers a generic fabric for constructing SOAs, its actual design and mapping to existing operational systems will differ dramatically depending on the situation under which it operates. To be sure, in the next few years, as SOAs become extensively deployed, more patterns are likely to emerge customized to novel implementation conditions.

Figure 10(a). UML communication diagram for a read operation

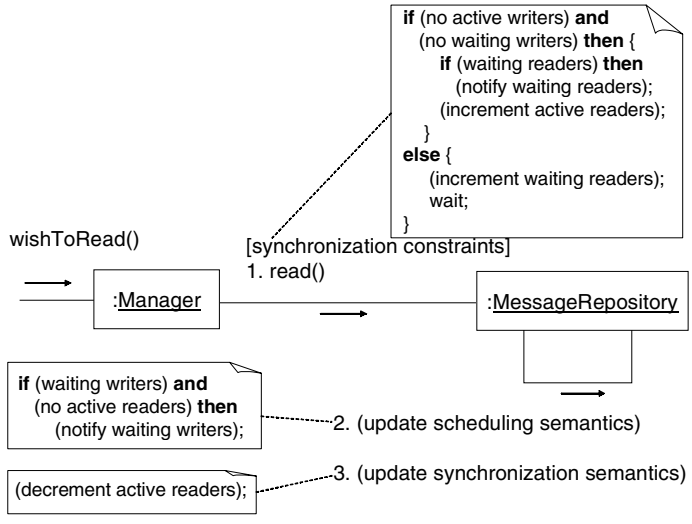
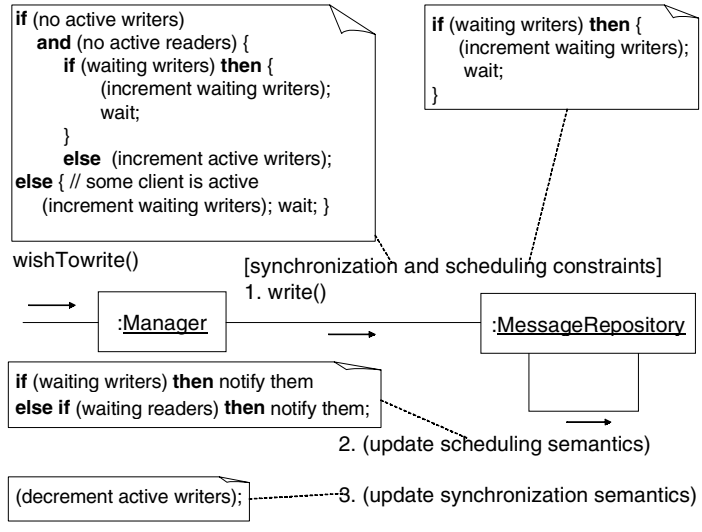


Figure 10(b). UML communication diagram for a write operation



In this chapter, we initiated the discussion on the use and collection of service patterns for the construction of efficient and effective SOAs. We anticipate that both software engineering researchers and practitioners will benefit from this discussion and will carry this work forward.

References

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A pattern language: Towns, buildings, construction*. Oxford: Oxford University Press.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine, *Computer Networks and ISDN Systems, 30*(1-7).

Burner, M. (2003, March). The deliberate revolution: Transforming integration with XML Web services. *ACM Queue, 1*(1).

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-V., & Irving, J. (1997, June 9-13). *Aspect oriented programming.* Proceedings of the 11th European Conference on Object-Oriented Programming (pp. 220–242), Jyväskylä, Finland.

Kreger, J. (2003). *Conceptual architecture for Web services*. Cambridge, MA: World Wide Web Consortium.

Larman, C. (2002). *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process* (2nd ed.). Prentice Hall.

Lea, D. (1999). *Concurrent programming in Java: Design principles and patterns* (2nd ed.). Addison-Wesley.

Monday, P. B. (2003). *Web services patterns: Java edition*. APress.

Nartovich, A., & Cunico, H. (2002). B2B e-marketplace with WebSphere Commerce v. 5.4. IBM Redbooks.

Papamarkos, G., Poulovassilis, A., & Wood, P. T. (2003, September 7-8). *Event-condition-action rule languages for the semantic Web.* In I. F. Cruz, V. Kashyap, S. Decker, & R. Eckstein (Eds.), Proceedings of the first International Workshop on Semantic Web and Databases (pp. 309–327), Humboldt-Universität, Berlin, Germany.

Roussos, G. (2004, April). *The Infomediator service pattern for mobile services* (Tech. Rep. No. BBKCS-04-05). Birkbeck: University of London, School of Computer Science and Information Systems.

Szyperski, C. (2002). *Component software: Beyond object-oriented programming* (2nd ed.). Addison-Wesley.

Wheeldon, R., & Levene, M. (2003, November 10-12). *The best trail algorithm for assisted navigation of Web sites*. Proceedings of the 1st Latin American Web Congress (pp. 166- 178), Santiago, Chile.

Section III

Mobile Services and Agents

Chapter XI

Concepts and Operations of Two Research Projects on Web Services and Mobile Web Services

Zakaria Maamar
Zayed University, United Arab Emirates

Abstract

Today, Internet technologies are enabling a wave of innovations that have an important impact on the way businesses deal with their partners and customers. Most businesses are moving their operations to the Web for more automation, efficient business processes, and global visibility. Web services are one of the promising technologies that help businesses in achieving these operations and being more Web-oriented. Besides the new role of the Internet as a vehicle of delivering Web services, a major growth in the field of wireless and mobile technologies is witnessed. Because users are heavily relying on mobile devices to conduct their operations, enacting Web services from mobile devices and possibly downloading these Web services for execution on mobile devices are avenues that academia and industry communities are pursuing. M-services denote the Web services in the wireless world. In this chapter, two research initiatives carried out at Zayed University are presented and referred to as SAMOS, standing for Software Agents for MObile Services, and SASC, standing for Software Agents for Service Composition.

Overview

Today, several businesses are adopting Web-based solutions for their operation, aiming for more process automation and more worldwide visibility. Thanks to the Web technology, users from all over the world can satisfy their needs by browsing and triggering the services of these businesses. Such services are usually referred to as Web services (Boualem, Zeng & Dumas, 2003). The advantages of Web services have already been demonstrated in various projects and highlight their capacity to be composed into high-level business processes. For example, a vacation business process calls for the collaboration of at least four Web services: flight reservation, accommodation booking, attraction search, and user notification. These Web services have to be connected with respect to a certain flow of control (first, flight reservation, then accommodation booking and attraction search). Multiple technologies are associated with the success of Web services, namely, WSDL (Web Services Definition Language), UDDI (Universal Description, Discovery, and Integration), and SOAP (Simple Object Access Protocol) (Curbera, Duftler, Khalaf, Nagy, Mukhi & Weerawarana, 2002). These technologies support the definition, advertisement, and binding of Web services.

Besides the Web expansion, we witness the tremendous progress in the field of wireless technologies. Telecom companies are deploying new services for mobile devices. Reading e-mails and sending messages between cell phones are becoming natural. Surfing the Web, thanks to the Wireless Application Protocol (WAP), is another evidence of the wireless technology development. The next stage (if we are not already in it) for telecom and IT businesses is to allow users to enact Web services from mobile devices and, possibly, to make these Web services runnable on mobile devices. M-services (M for mobile) denote these new type of Web services (Maamar & Mansoor, 2003).

It is accepted that composing multiple services (whether Web services or M-services) rather than accessing a single service is essential. Berardi et al. (2003) report that composition addresses the situation of a client's request that cannot be satisfied by any available service, whereas a composite service obtained by combining a set of available services might be used. Searching for the relevant services, integrating these services into a composite service, triggering the composite service, and monitoring its execution are among the operations that users will be in charge of. Most of these operations are complex, although repetitive, with a large segment suitable for computer aids and automation. Therefore, software agents are deemed appropriate candidates to assist users in their operations (Jennings, Sycara & Wooldridge, 1998).

Throughout this chapter, two research initiatives that our research group is conducting at Zayed University are presented. These initiatives are respectively **SAMOS**, standing for **Software Agents for MObile Services**, and **SASC,** standing for **Software Agents for Service Composition**. Both initiatives deal with the composition of services using software agent-oriented approaches. This chapter is structured as follows. The Background section outlines the concepts that are used in our research work, such as mobile computing and software agents. The next section overviews some research projects related to mobile computing. The SAMOS Research Initiative and SASC Research

Initiative sections present SAMOS and SASC in terms of architecture, types of agents, and operation. In the last section, we draw our conclusions.

Background

Mobile Computing

Mobile computing refers to systems in which computational components, either hardware or software, change locations in a physical environment. The ability to move from one location to another is because of the progress in several technologies: component miniaturization, wireless networks, and mobile-code programming languages. Categories of mobility include (Wand & Chunnian, 2001): hardware mobility, software mobility, and combined mobility. A code that is downloaded from a server to a mobile phone combines both hardware and software mobility. The Overview of Some Research Projects Related to Mobile Computing section provides more details on mobile computing using research projects as examples.

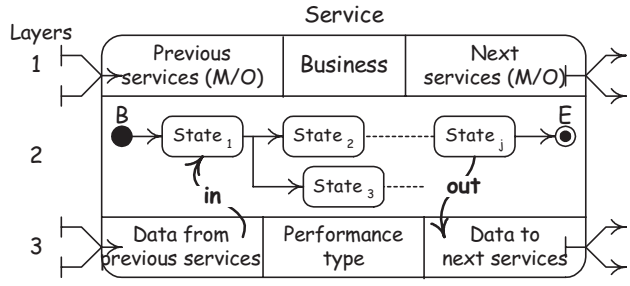
Web Services and M-Services

A Web service is an accessible application that can be automatically discovered and invoked by other applications and humans. An application is a Web service if it is (Benatallah et al., 2003): (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed mainly for interorganizational situations rather than for intraorganizational situations; and (iii) easily composable so its composition with other Web services does not require the development of complex adapters.

Two definitions are associated with an M-service (Maamar & Mansoor, 2003). The weak definition is to remotely trigger a Web service for execution from a mobile device. In that case, the Web service acts as an M-service. The strong definition is to wirelessly transfer a Web service from its hosting site to a mobile device where its execution happens. In that case, the Web service acts as an M-service that is: (i) transportable through wireless networks; (ii) composable with other M-services; (iii) adaptable with regard to the computing features of mobile devices; and (iv) runnable on mobile devices. In both SAMOS and SASC initiatives, only the M-services that comply with the strong definition are considered.

The differences between Web services and M-services are depicted at two levels. The first level concerns the communication medium (wired channel for Web services versus wireless channel for M-services). And the second level concerns the location of where the processing of the service occurs (server side for Web services versus user side for M-services).

Figure 1. Service chart diagram of a component service



In Maamar, Benatallah, and Mansoor (2003), we introduced the concept of service chart diagram as a technique for modeling and specifying the component services that participate in composite services. A service chart diagram enhances the state chart diagram of UML. In fact, the emphasis this time is on the context surrounding the execution of a service rather than only on the states that a service takes (Figure 1).

A service chart diagram wraps the states of a service into five perspectives, each perspective has a set of parameters. The state perspective corresponds to the state chart diagram of the service. The flow perspective corresponds to the execution chronology of the composite service in which the service participates (Previous services/Next services parameters; M/O respectively stands for Mandatory and Optional). The business perspective identifies the organizations (that is, providers) that make the service available (Business parameter). The information perspective identifies the data that are exchanged between the services of the composite service (Data from previous services/Data for next services parameters). Because the services participating in a composition can be either mandatory or optional, the information perspective is tightly coupled to the flow perspective with regard to mandatory data and optional data. Finally, the performance perspective illustrates the ways the service can be invoked for execution (Performance type parameter).

Software Agents

A software agent is a piece of software that autonomously acts to undertake tasks on behalf of users (Jennings et al., 1998). The design of many software agents is based on the approach that the user only needs to specify a high-level goal instead of issuing explicit instructions, leaving the how and when decisions to the agent. A software agent exhibits a number of features that make it different from other traditional components including autonomy, goal orientation, collaboration, flexibility, self-starting, temporal continuity, character, communication, adaptation, and mobility. It is noted that not all of these characteristics have to embody an agent.

Besides the availability of several approaches and technologies related to the deployment of Web services (for example, SOAP, UDDI, Salutation), they are all tailored to a context of type wired. In a similar context, all the computing resources are fixed and connected through a permanent and reliable communication infrastructure. The application of these approaches and technologies to a context of type mobile computing is not straightforward. Indeed, major adjustments are required because of multiple obstacles ranging from potential disconnections of mobile devices and unrestricted mobility of persons to power scarcity of mobile devices and possibility of capturing the radio signals while in the air. These obstacles highlight the suitability of software agents as potential candidates to overcome them. First, an agent is autonomous. Thus, it can make decisions on the user's behalf while this one is disconnected. Second, an agent can be mobile. Thus, it can move from one host to another. Continuous network connectivity is not needed. Third, an agent is collaborative. Thus, it can work with other agents that identify, for example, the providers of Web services. Last but not least, an agent is reactive. Thus, it can monitor the events that occur in the user's environment, so relevant actions can be promptly taken.

Overview of Some Projects Related to Mobile Computing

There exist several research projects that have studied how mobile devices can change the way of doing business and undertaking operations. In HP Laboratories, the authors in Milojicic et al. (2001) worked on delivering Internet services to mobile users. This work was conducted under the project Ψ for Pervasive Services Infrastructure (PSI). The Ψ vision is "any service to any client (anytime, anywhere)". The project investigated how offloading parts of applications to midpoint servers can enable and enhance service execution on a resource-constrained device.

The Odyssey project aimed at providing system support for mobile and adaptive applications (Noble et al., 1997). Odyssey defined a platform for adaptive mobile data access on which different applications, such as Web browser, video player, and speech recognition, can run on top. The Odyssey approach is to adjust the quality of accessed data to match available resources.

Ninja aimed at suggesting new types of robust and scalable distributed Internet services (Ninja, 2001). The objective in Ninja is to meet the requirements of an emerging class of extremely heterogeneous devices that would access these services in a transparent way. In Ninja, the architecture considered four elements: bases, units, active proxies, and paths. Proxies are transformational intermediaries that are deployed between devices and services to shield them from each other. A service discovery service is also suggested in Ninja for two reasons: (i) enable services to announce their presence and (ii) enable users and programs to locate the announced services.

SAMOS Research Initiative

In addition to the role of the Internet as a vehicle of provisioning Web services, it is noticed that more Web services will be delivered to people who use mobile devices and, particularly, to those who are on the move most of the time (for example, sales representatives). It is also noticed that mobile devices are being enhanced with extra computing resources and advanced functionalities (Yunos, Gao & Shim, 2003). Unfortunately, the growth in the development and use of mobile devices is subject to multiple challenges. For instance, mobile devices are still bound to their batteries for operation, which leads to limit, to a certain extent, their computation performance.

It occurs that mobile users have to postpone their operations because they lack appropriate facilities running on their mobile devices (for example, an application that converts a drawing file into a format that the user's mobile device can display). In SAMOS, we support such users by allowing them: (i) to search for additional facilities, when needed; (ii) to fetch these facilities to their mobile devices; and (iii) to conduct these two operations in a transparent way. Various solutions are put forward to handle these points and are discussed throughout this part of the chapter. A solution to point (i) consists of devising brokering mechanisms. A solution to point (ii) consists of using wireless communication channels. Finally, a solution to point (iii) consists of using Software Agents (SAs) to make the search for and fetch the facilities transparent to users.

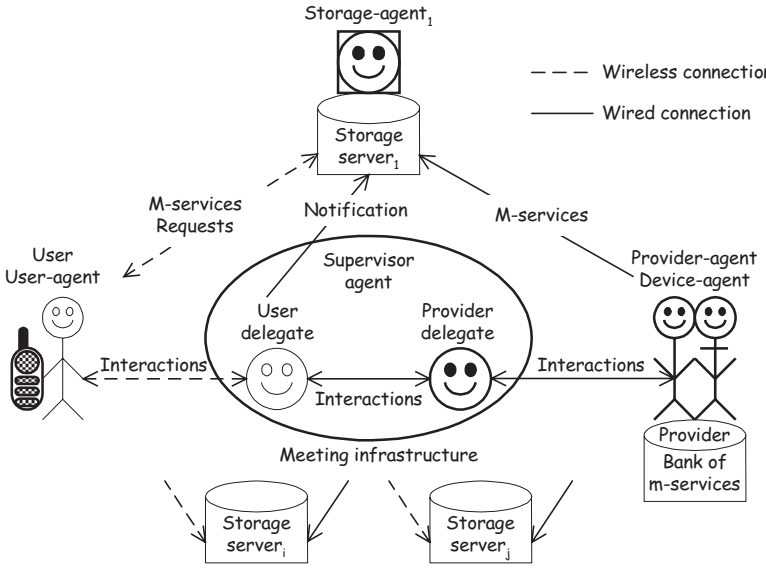
Architecture and Software Agents of SAMOS

Brokering mechanisms and SAs are considered in the design and development of SAMOS. The salient features of the architecture of SAMOS are:

- Three types of SAs: user-agent, provider-agent, and device-agent. The first type is associated with users of M-services, whereas the second and third types are associated with providers of M-services.
- A software platform, called Meeting Infrastructure (MI), is headed by a supervisor-agent. This MI has a brokering role (Maamar, Dorion & Daigle, 2001).
- Two types of delegates, namely, provider-delegate and user-delegate. Delegates respectively interact on behalf of user-agents and provider-agents in the MI.
- Storage servers that save the sequence of M-services to be submitted to mobile devices for execution. Storage servers are spread across networks, and storage-agents are responsible for managing these servers. In SAMOS, a sequence corresponds to a composite service that has M-services as primitive components.

Figure 2 illustrates the architecture of SAMOS. It consists of four parts: user, provider, MI, and storage. The MI and storage parts are wirelessly linked to the user component, whereas the MI and storage parts are linked to the provider component with wires.

Figure 2. Architecture of SAMOS



The user part consists of users and user-agents. User-agents accept users' needs, convert them into requests, and submit them to user-delegates. The supervisor of the MI creates user-delegates on requests from user-agents. To satisfy users' requests, user-delegates interact with provider-delegates.

The provider part consists of providers, provider-agents, and device-agents. Provider-agents act on behalf of providers by advertising their M-services to user-delegates through provider-delegates. Plus, provider-agents monitor the behavior of providers when new M-services are offered and, thus, need to be announced. In Figure 2, M-services are gathered into a bank on which provider-agents and device-agents reside. Provider-agents create provider-delegates. In the provider part, device-agents support the work of provider-agents, whereas the role of device-agents is to wrap the M-services before they are sent to mobile devices for execution. The rationale of device-agents is to consider the differences that exist between mobile devices (for example, screen size, processor power).

The MI part is a software platform in which user-delegates and provider-delegates interact in a local and secure environment (Maamar et al., 2001). In an open environment, most of the interactions occurring between requesters of services and providers of services are conducted through third parties (referred to as brokers). Despite its important role, a third party can easily become a bottleneck. To overcome this problem, requesters and providers need a common environment in which they meet and interact directly. The MI corresponds to this common environment. In SAMOS, the supervisor-

agent of the MI has several responsibilities including monitoring the interactions that occur within the MI and making the MI a safe environment.

The storage part receives the sequence of M-services that will be submitted to mobile devices for performance. In SAMOS, one of the operation principles is to submit the M-services to mobile devices for execution one at a time. This restriction is due to the limited resources of these devices. However, the restriction can be handled (that is, adjust the number of M-services to be submitted) based on the computing resources of a mobile device and the bandwidth of the wireless communication channels. Several advantages are obtained from the use of storage servers. For instance, a user-agent does not have to deal with several providers. Its unique point of contact for getting the M-services is the storage-agent. The same thing applies to device-agents that will only be interacting with few storage-agents instead of multiple user-agents. Security is increased for both users and providers. Indeed, storage servers are independent platforms where security controls are carried out.

User-Oriented Components

A user-agent resides in a mobile device. First, the user interacts with the user-agent to arrange requests. After submitting those requests to the user-delegate, the user-agent takes a standby state and waits for notifications from its user-delegate. Notifications concern the sequence of M-services that satisfies the user's requests. Before executing them on the user's device, the M-services are put in a storage server. The MI supervisor-agent suggests to the user-delegate the storage server to be used based, for example, on the server's location. To download the M-services one at a time from the storage server to the user's device, the user-agent communicates with the storage-agent. The user-agent keeps track of the execution of the M-services before it asks the storage-agent to submit further M-services. When an M-service is received, the executed M-service is deleted from the mobile device. Finally, the user-agent informs the user about the completed requests.

A user-delegate resides in the MI, acting on behalf of the user-agent. The user-delegate receives the user's requests from the user-agent. Afterward, it interacts with provider-delegates. The purpose of these interactions is to match the requests of users to the M-services of providers that are announced. In case there is a match (we assume that there is always a match), the user-delegate designs the sequence of M-services that satisfies the user's requests. Information about this sequence is sent afterward to the storage-agent. The objective is to make the storage-agent ready for receiving the M-services from device-agents. Furthermore, the user-delegate notifies the user-agent about the sequence of M-services it has prepared for its user. To set up a sequence, the storage-agent knows the M-service that comes before and after the M-services to be submitted by a device-agent (flow perspective of a service chart diagram, Figure 1). Instead of creating a user-delegate on a mobile platform and shipping that delegate to the MI, we suggested to perform this operation in the MI for two main reasons: (i) even if we expect a major improvement in the resources of mobile devices, those resources have to be used in a *rationale* way and (ii) the wireless connection that transfers the user-delegate to the MI is avoided.

Provider-Oriented Components

A Provider-agent resides in a provider site running on top of its resources such as M-services. Provider-delegates broadcast the M-services to user-agents through user-delegates. The provider-agent is in constant interaction with its provider-delegate. For instance, it notifies the provider-delegate about the negotiation strategy it has to follow with user-delegates.

A device-agent resides in a provider site. Its responsibility is to wrap the M-services according to the features of the devices to which these M-services will be submitted for performance. Initially, the M-services are sent to storage servers. The provider-agent has already submitted the contact details of the storage server to the device agent. We recall that the user-delegate informs the storage-agent of the storage server about the M-services it will receive. Double checking the information that user-delegates and provider-delegates submit to a storage-agent offers more security to the agents of SAMOS.

A provider-delegate resides in the MI, acting on behalf of a provider-agent. In SAMOS, the provider-delegate is responsible for interacting with user-delegates regarding the M-services it offers. In addition, the provider-delegate interacts with its provider-agent for notification purposes. Notifications are then forwarded to device-agents for action. We recall that the provider-agent is responsible for creating the provider-delegate and its transfer to the MI.

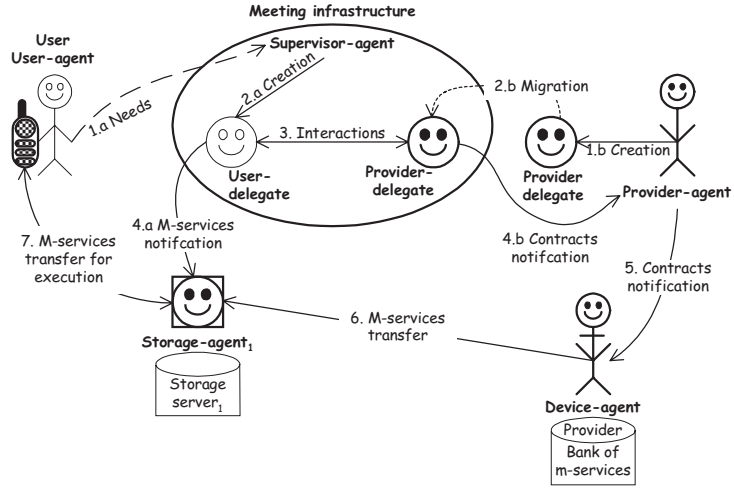
MI-Oriented Components

The supervisor-agent resides in the MI and has several responsibilities: it supervises the operations that occur in the MI; it mediates in case of conflicts between user-delegates and provider-delegates; it sets user-delegates and assigns them to user-agents; it checks the identity of provider-delegates when they arrive from provider sites; and finally, it suggests to user-delegates the storage server to be used.

Storage-Oriented Components

A Storage-agent runs on top of a storage server. This server saves the M-services to be sent to mobile devices for performance. According to the information on the sequence of M-services it receives from the user-delegate, the storage-agent arranges the sequence as the M-services start arriving from providers. As soon as this sequence is completed, it notifies the user-agent in order to get ready for receiving the M-services. Based on the requests it receives from the user-agent, the storage-agent submits the M-services one at a time. These M-services are ready for execution. The deletion of M-services from the storage servers and mobile devices follows certain reliability rules. These rules ensure that the M-services to be sent to a mobile device for execution are successfully received and executed.

Figure 3. Operation of SAMOS



Operation of SAMOS

The operation of SAMOS consists of five stages (Figure 3): agentification, identification, correspondence, notification, and realization. The purpose of the agentification stage is to set up the different infrastructures and agents that constitute SAMOS. User-agents are established at the user level. Provider-agents and device-agents are established at the provider level, too. Finally, the meeting infrastructure and storage servers, including their storage-agent, are deployed. In Figure 3, operations (1.a) and (1.b) illustrate the agentification stage.

The purpose of the identification stage is to inform the supervisor-agent of the MI about the existence of users and providers who are interested in using SAMOS. At the agentification stage, user-agents and provider-agents are respectively installed on top of mobile devices of users and resources of providers. The outcome of the identification stage is the creation of user-delegates and the reception of provider-delegates arriving from provider-sites. Creation and reception operations occur in the MI. Provider-agents notify the supervisor-agent about their readiness to submit the provider-delegates to the MI. User-agents inform the supervisor-agent about the users' requests they would like to submit. In Figure 3, operations (2.a) and (2.b) illustrate the identification stage.

The purpose of the correspondence stage is to enable user-delegates and provider-delegates to get together. In Figure 3, operation (3) illustrates this stage. User-delegates

have requests to satisfy, and provider-delegates have services to offer. First, the user-delegate searches for the provider-delegates that have the M-services it needs. Two approaches are offered (Maamar & Mansoor, 2003):

a) The user-delegate asks the supervisor-agent to suggest a list of provider-delegates that have the services it needs.

b) The user-delegate requests from the supervisor-agent the contact details of all the provider-delegates that exist in the MI.

Independently of the approach that is adopted, the user-delegate submits its needs of services to a shortlist of selected provider-delegates. Based on different parameters, such as workload and commitments, provider-delegates answer the user-delegate. At this time of our research in SAMOS, it is assumed that providers do not have services in common. Consequently, there is no need for a user-delegate to look for the best service. Once the user-delegate and provider-delegates agree on the M-services to use, notifications are sent to different recipients as it is discussed in the next stage.

The purpose of the notification stage is to inform different agents about the agreements between user-delegates and provider-delegates. In Figure 3, operations (4.a), (4.b), (5), and (6) illustrate this stage. Regarding the user-delegate, it is in charge of informing (i) the user-agent about the sequence of M-services it has established to satisfy its user's request and (ii) the storage-agent about the sequence of M-services it will receive from different device-agents. Regarding the provider-delegate, it notifies the provider-agent about its agreements with a user-delegate. Based on the information it receives from its provider-delegate, the provider-agent forwards this information to the device-agent. This information is about the M-services that are involved and the storage server that is used. Among the actions the device-agent takes is to submit the M-services to the storage-agent of the storage server.

The purpose of the realization stage is to execute the sequence of M-services that the user-delegate has designed. User-agent and storage-agent participate in this stage. We recall that the user-delegate has already informed the storage-agent about the M-services it will receive from device-agents. Before the user-agent starts asking the storage-agent for the M-services it has, it waits for a notification message from the storage-agent mentioning that the sequence is ready for submission and, thus, for execution. In Figure 3, operation (7) illustrates the realization stage. In the realization phase, reliability is one of the concerns that have been considered in SAMOS. We consider a storage server as a backup server for the M-services. When a storage-agent sends an M-service to a user-agent, the storage-agent keeps a copy of this service at its level. The storage-agent deletes that M-service when the user-agent asks for the M-service that follows the one it has received. For the last M-service of a sequence, the user-agent sends an acknowledgment message to the storage-agent, so this M-service can be deleted.

Summary on SAMOS

In this part of the chapter, we discussed the use of M-services in the context of SAMOS. M-services are seen as a logical extension to the widespread use of Web services in the wireless world. Considering mobile devices as computing platforms is becoming a reality as the networks that make them reachable are in constant progress, offering more bandwidth and ensuring more reliability and efficiency. For instance, third-generation communication systems are providing high quality streamed Internet content (Chisalita & Shahmehri, 2001). In addition to higher data rates, these systems back the provision of new value-added services to users, such as geographical positioning and mobile payment.

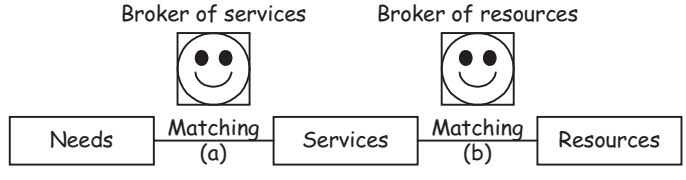
SASC Research Initiative

Despite that provisioning Web services is a very active area of research and development, very little has been done to date regarding their integration with M-services. Several obstacles still exist including throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections.

A framework that composes services, whether Web services or M-services, should offer more opportunities to users to conduct operations regardless of (i) the type of services, (ii) the location of users, and (iii) the computing resources on which services will be performed. This situation is challenging due to the gap existing between wired and wireless. First, Web services are associated with fixed devices. However, M-services are associated with mobile devices. Second, the execution of Web services occurs in the server side, whereas the execution of M-services occurs in the client side (according to the strong definition of what an M-service is). Third, fixed devices are not resource-constrained which is not the case for mobile devices. Despite the multiple opportunities that could be offered to users, few research efforts are being dedicated to the composition of Web services and M-services.

Because the information space is already full of several providers of services, a broker that matches services to needs of users is one step in the design of the SASC framework

Figure 4. Needs versus services and services versus resources



(Figure 4a). On the other side, because services require resources on which they can be computed, there is a need for another broker as a second step in the design of the SASC framework (Figure 4b). This broker matches services (those that satisfy users' needs) to the resources of providers.

In the previous paragraph, it is shown that two types of providers are involved: provider of services and provider of resources (a provider can play both roles). Due to this distinction of providers, a user with a fixed or mobile device is also seen in the SASC framework as a provider of resources in the composition framework (that is, users' devices are advertised to the broker of resources). Considering users as providers of resources enables them to play an active role instead of always being limited to their traditional passive role of consumers. The rationale is to take advantage of the spare resources that are available on devices. It is observed that many of the systems are often underutilized due to geography factor. Busy hours in one time zone tend to be idle hours in another zone. Therefore, demands for computational resources can be met with hosts that have idle resources. For the needs of the SASC initiative, the term composite service denotes the set of component services (whether composite services, Web services, or M-services) that take part in a composition.

Architecture and Software Agents of SASC

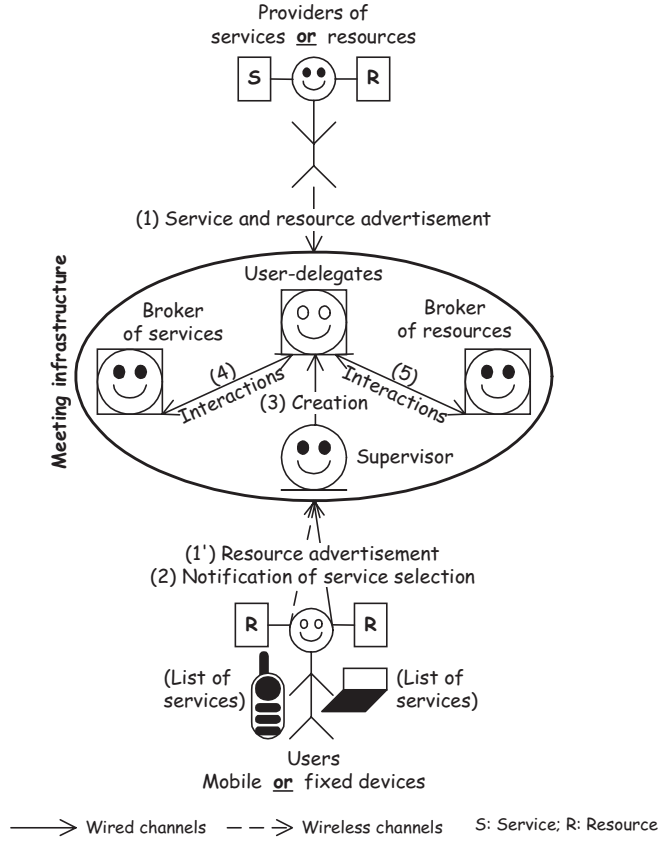
Figure 5 is the agent-based architecture upon which the SASC framework is deployed. The architecture has three parts. The first part corresponds to providers of services (S) or resources (R). The second part corresponds to consumers of services (that is, users) with their fixed or mobile devices. Finally, the third part corresponds to the meeting infrastructure (similar to the one that is used in SAMOS) on which brokers carry out the matching operations between needs of users and services of providers and, later on, between services of providers and resources of providers (Figure 4). The meeting infrastructure connects the provider and consumer parts. To keep Figure 5 clear, the different agents that populate the architecture are not represented. The core agents of the framework are briefly described below.

Provider-agents are specialized into two types: resource-provider-agents and service-provider-agents. Resource-provider-agents handle the execution of the services of service-provider-agents. In the MI, resource-delegates and service-delegates, respectively, represent resource providers and service providers (delegates are agents but are given a different name to avoid confusion).

User-agents reside in the devices of users and are specialized into two types: fixed-user-agents (for users of fixed devices) and mobile-user-agents (for users with mobile devices). In the meeting infrastructure, user-delegates represent users to whom their needs are submitted.

Broker-agents are specialized into two types (Figure 4): service-broker-agent and resource-broker-agent. A service-broker-agent receives (i) notifications from service-delegates about their offers of services and (ii) requests from user-delegates about their needs of services. Whereas a resource-broker-agent receives (i) notifications from

Figure 5. Architecture of SASC



resource delegates regarding their respective offers of resources and (ii) requests from service-delegates regarding their needs of resources.

The supervisor-agent is in charge of the MI. For instance, it creates user-delegates and checks the security credentials of service-delegates and resource-delegates once both arrive from their original host. It should be noted that the security of delegates is beyond the scope of this chapter. However, the security of the services that run on computing resources is discussed in Maamar, Hamdi, Mansoor, and Bhati (2003).

Rationale of User/Resource/Service-Delegates

- Because mobile devices are resource-constrained, several authors, such as Jailani, Othman, and Latih (2002) and Messer, Greeberg, Bernadat, and Milojicic (2002),

observe that it is appropriate to offload computing from mobile devices to fixed ones. In SASC, once the service-broker-agent matches users' needs to providers' services, the next step for a user-delegate is to integrate the component services into a composite service. This integration requires resources that have to be used in a *rationale* way when it comes to mobile devices. Therefore, it is preferable to undertake the development of composite services in the MI rather than in mobile devices. In addition, it may happen that the user-delegate needs further information from a broker to complete its work on a composite service. Since the user-delegate already resides in the MI, it locally interacts with the broker. This constitutes another argument in favor of using user-delegates. Because of the advantages that local interactions offer, even users of fixed devices are encouraged to develop their composite services in the meeting infrastructure.

- When there is a match between the needs of a user and the offers of services, the service-broker-agent locally notifies the user-delegate and remotely notifies the relevant service-provider-agents. Since remote exchanges are subject to obstacles (for example, network reliability, transfer safety), providers of services are associated with service-delegates. Service-delegates are transferred from the sites of their respective provider-service-agents to the MI. After the first match is over, the service-delegate informs the resource-broker-agent about its needs of resources; certain services have been selected and, thus, need to be executed. Once the resources are identified, the service-delegate remotely interacts with the resource-provider-agents about the modalities of using their resource. Similarly to service-delegates, it is more convenient if the interactions between service-delegates and resource-provider-agents occur locally. Therefore, resource-provider-agents have resource-delegates to act on their behalf in the meeting infrastructure.

Operation of SASC

The operation of SASC consists of six stages: initialization, advertisement, search for services, search for resources, refinement, and completion. Below is a summary of the main actions that occur in each stage.

Initialization stage:

- Agentify users and providers.
- Create supervisor and brokers and deploy them in the meeting infrastructure.- Embody agents with operation mechanisms.

Advertisement stage:

- Create service-delegates and resource-delegates.
- Transfer delegates to the meeting infrastructure.

- Check delegates before they enter the meeting infrastructure.
- Advertise services and resources to brokers.

Search for services stage:

- Create user-delegates in the meeting infrastructure.
- Submit users' needs to user-delegates.
- Interact with service-broker-agent.
- Match user's needs with providers' services.
- If positive match, return list of service-delegates to user-delegate.

Search for resources stage:

- Interact with resource-broker-agent.
- Match selected service with providers' resources.
- If positive match, return list of resource-delegates to service-delegates.
- Select a specific resource-delegate for a service.
- Transfer service for execution to resource-delegate site.

Refinement stage:

- Combine outcomes of search for services and search for resources stages.
- Submit new details (version and processing type) on service to user-delegate.
- Finalize selection of service-delegate by user-delegate.

Completion stage:

- Work on next service based on details of previous service.
- Select a specific resource-delegate for a service.
- Transfer service for execution to resource-delegate site.
- Submit new details (version and processing type) on service to user-delegate.
- Finalize selection of service-delegate by user-delegate.Keep running completion stage until all services are processed.

The purpose of the initialization stage is to perform the agentification of the components of SASC (that is, provider and user). Each provider/user is associated with an agent that exhibits a behavior in terms of resources to have, services to offer, and needs to satisfy. User-agents and provider-agents are respectively installed on top of users' devices and providers' resources/services. Moreover, further agents (supervisor, service-broker, and resource-broker) are created in the MI. Afterwards, information on "services versus

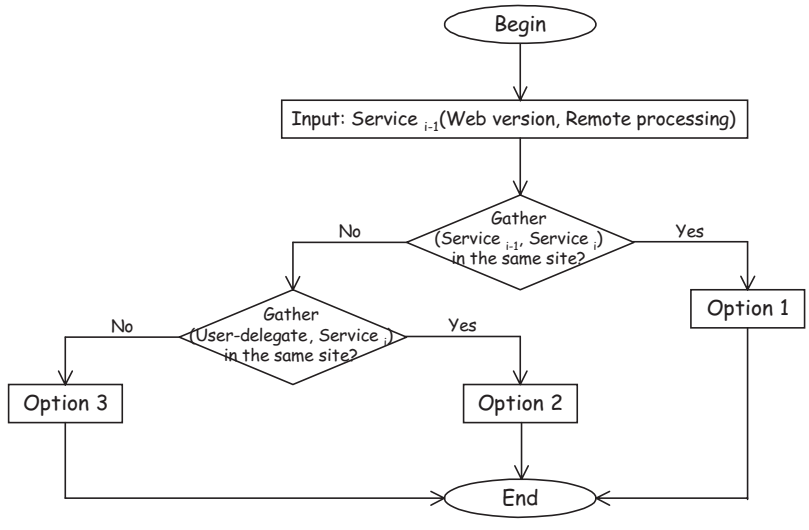
needs" is loaded into the knowledge base of the service-broker-agent (operation done by the administrator of SASC). Likewise, information on "resources versus services" is loaded into the knowledge base of the resource-broker-agent. Finally, the supervisor-agent is embodied with the mechanisms of creating user-delegates as well as verifying and installing service-delegates and resource-delegates.

The purpose of the advertisement stage is to notify the brokers about the available services and resources that are made available to the user community. As a first step, service/resource-provider-agents create service/resource-delegates and transfer them to the MI. Because mobile devices are resource-constrained, the supervisor-agent creates the resource-delegates on behalf of the users of these devices. Mechanisms that embody a service/resource-delegate are several, including how to announce itself to the supervisor-agent, how to register at the service/resource-broker-agent, and how to notify its respective service/resource-provider-agent. When service/resource-delegates access the meeting infrastructure, they register at the appropriate broker to submit their offers of services/resources. It should be noted that service-delegates have a dual role (Figure 4): (i) as a provider of services when they interact with the service-broker-agent and (ii) as a consumer of resources when they interact with the resource-broker-agent. The purpose of the search for services stage is to look for the services that satisfy a user's needs. On reception of the needs, the supervisor-agent creates a user-delegate to be in charge of user satisfaction. First of all, the user-delegate interacts with the service-broker-agent. The purpose is to identify the services of service-delegates that satisfy the user's needs. In case certain services are identified, the service-broker-agent notifies the user-delegate and the service-delegate of these services. Because service-delegates may have services in common, the user-delegate has to select a particular service-delegate. However, the user-delegate delays its selection until further details on services are provisioned. These details concern the cost, version, and processing type of each service.

The purpose of the search for resources stage is to identify the resources that support the execution of the services (that is, those that have been identified in the search for services stage). In the MI, service-delegates trigger the matching between services and resources. The identification of the resources is conducted service per service. As it will be described, the selection of a resource for any service depends on the version and type of processing (that is, remote processing or local processing) of the direct predecessor service of this service. On receiving the service-delegates' requests, the resource-broker-delegate identifies the appropriate resource-delegates. Since several resource-delegates can support the execution of the same service, a service-delegate has to select a resource-delegate. In SASC, the selection strategy consists of minimizing the cost of running a service on a resource considering the version and type of processing of this service. At this time of the operation of SASC, each service-delegate knows exactly for its service the version and type of processing to offer to the user-delegate.

The purpose of the refinement stage is to improve the outcome of the search for services stage. Since a service-delegate is aware of the version and type of processing of the service it will offer to the user-delegate, the service-delegate prepares a cost for that service. In its offer, the service-delegate includes the cost of running the service on a

Figure 6. Application of location criterion to service selection



resource. After it receives all the offers from service-delegates, the user-delegate selects for a service a particular service-delegate. The user-delegate minimizes the cost of getting the service from all the service-delegates. When the user-delegate selects a service-delegate, this service-delegate submits to the resource-provider-agent the following details: (i) the service this resource-provider-agent will receive for processing; (ii) the version of this service; (iii) the user-delegate that will trigger the processing of this service; and (iv) the way this service will be invoked for processing.Completion is the final stage in the operation of SASC. Here, the selection of any service directly depends on the version and type of processing of its direct predecessor service. In addition to the cost criterion that was used in the previous stages, another selection criterion is now included, namely, location. Location criterion aims at gathering the maximum number of services for execution in the same computing site[1]. By computing site, it is meant: location of resource-provider-agents and current location of the user-delegate. By gathering services in the same computing site, the following advantages are obtained: (i) extra moves of the user-delegate to distant sites of resource providers are avoided and (ii) extra remote communication and data exchange messages between user-delegates and resource-provider-agents are avoided, too. Therefore, the location criterion is privileged over the cost criterion. When the details on a service are known, the user-delegate requests from the service-delegates to identify the resource-delegates for the next service. The work on service$_{(i)}$ is decomposed into three cases: Web version and remote processing of service$_{(i-1)}$, Web version and local processing of service$_{(i-1)}$, and M-version and local processing of service$_{(i-1)}$. To keep the chapter self-contained, only the first case is presented.

Web Version and Remote Processing of Service$_{(i-1)}$ Case

Since the processing of the Web version of service$_{(i-1)}$ has been remotely conducted, this means that the user-delegate is in a different site to the execution site of service$_{(i-1)}$. Three exclusive options are offered to the user-delegate to make a decision on service$_{(i)}$ (Figure 6).

- Option 1: the processing of service$_{(i)}$ takes place in the site of service$_{(i-1)}$ in order to comply with the location criterion. Therefore, the user-delegate requests from the service-delegates of service$_{(i)}$ to check with the resource-broker-agent what follows: does resource-delegate$_{(i-1)}$ support the remote processing of the Web version of service$_{(i)}$? If yes, then the service-delegates have to select resource-delegate$_{(i-1)}$. Afterwards, the user-delegate selects a service-delegate based on the cost criterion. As a result, the Web versions of service$_{(i)}$ and service$_{(i-1)}$ will be both installed in site$_{(i-1)}$ of resource-delegate$_{(i-1)}$. The user-delegate will remotely process them.
- Option 2: the processing of service$_{(i)}$ takes place in the site of the user-delegate in order to comply with the location criterion. Option 2 exists because resource-delegate$_{(i-1)}$ does not support the remote processing of the Web version of service$_{(i)}$. Therefore, the user-delegate requests from the service-delegates of service$_{(i)}$ to check with the resource-broker-agent what follows: does the resource-delegate of the current site of the user-delegate support the local processing of the Web version of service$_{(i)}$? If yes, then the service-delegates have to select resource-delegate$_{(i-1)}$. Afterwards, the user-delegate selects a service-delegate based on the cost criterion. As a result, the Web version of service$_{(i-1)}$ and the Web version of service$_{(i)}$ will be located in different sites. However, the user-delegate will locally process service$_{(i)}$.
- Option 3: the processing of service$_{(i)}$ takes place in any site (different from the site of the user-delegate and the site of service$_{(i-1)}$). Option 3 happens because the site of the user-delegate does not support the local processing of the Web version of service$_{(i)}$. In that case, the location criterion does not hold. Search for services and search for resources stages as previously described are carried out in order to define the version and type of processing of service$_{(i)}$ and the respective resource-delegate.

Summary on SASC

Future computing environments will involve a variety of devices with different capacities in terms of processing power, screen display, input facilities, and network connectivity. Furthermore, a variety of services will be offered to users making the use of these devices important in their performance. In this second part of the chapter, we presented SASC that aims at composing services whether Web services or M-services. The backbone of

SASC is a software agent-based architecture that integrates several agents such as user, provider, service, and resource. SASC also aims at provisioning services independently of the location of users and the resources they may be using. Service provisioning has relied on two selection criteria (execution cost and resource location) to identify which resources should be assigned to which services.

Conclusion

In this chapter, we presented the research initiatives that are carried out @ Zayed University on Web services and Mobile Web services. Among these initiatives, we cited SAMOS, standing for Software Agents for MObile Services, and SASC, standing for Software Agents for Service Composition. New issues that are related to mobile Web services and their integrations with traditional Web services are raised, varying from low bandwidth and high latency of wireless networks to screen sizes of mobile devices. To deal with these issues, software agents are considered due to their various features. For instance, a software agent is autonomous. Thus, it can make decisions on the user's behalf while this one is disconnected. Second, a software agent can be mobile. Thus, it can move from one host to another. Continuous network connectivity is not needed. The major progress happening in the wireless field will be offering the right mechanisms to users to conduct their daily activities over a variety of mobile devices. Three major factors should boost the penetration and expansion of mobile Web services, namely: personalization, time-sensitivity, and context-awareness.

Acknowledgments

The author would like to thank the referees for their valuable comments and suggestions of improvements. The author also acknowledges the contributions of Q. H. Mahmoud (Guelph University, Canada) and W. Mansoor (Zayed University, U.A.E.) to SAMOS, and B. Benatallah (University of New South Wales, Australia) and Q. Z. Sheng (University of New South Wales, Australia) to SASC.

References

Benatallah, B., Sheng, Q. Z., & Dumas, M. (2003, January/February). The SELF-SERV environment for Web services composition. *IEEE Internet Computing, 7*(1).

Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Mecella, M. A. (2003). *Foundational vision for e-services.* Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web (WES'2003) in conjunction with The 15th

Conference On Advanced Information Systems Engineering (CAiSE'2003), Klagenfurt/Velden, Austria.

Chakraborty, D., Perich, F., Joshi, A., Finin, T., & Yesha, Y. (2002). *A reactive service composition architecture for pervasive computing environments.* Proceedings of the 7th Personal Wireless Communications Conference (PWC'2002), Singapore.

Chisalita, I., & Shahmehri, N. (2001). *Issues in image utilization with mobile e-services.* Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2001), Boston, Massachusetts.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002, March/April). Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing, 6*(2).

Jailani, N., Othman, M., & Latih, R. (2002). *Secure agent-based marketplace model for resource and supplier broker.* Proceedings of the 2nd Asian International Mobile Computing Conference (AMOC'2002), Langkawi, Malaysia.

Jennings, N., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, *1*(1).

Maamar, Z., Dorion, E., & Daigle, C. (2001, December). Towards virtual marketplaces for e-commerce. *Communications of the ACM, 44*(12).

Maamar, Z., Benatallah, B., & Mansoor, W. (2003). *Service chart diagrams - Description & application.* Proceedings of the 12th International World Wide Web Conference (WWW'2003), Budapest, Hungary.

Maamar, Z., Yahyaoui, H., Mansoor, W., & Bhati, A. (2003). *Towards an environment of mobile services: Architecture and security.* Proceedings of the 2003 International Conference on Information Systems and Engineering (ISE'2003), Montreal, Canada.

Maamar, Z., & Mansoor, W. (2003). Design and development of a software agent-based and mobile service-oriented environment. *e-Service Journal, Indiana University Press, 2*(3).

Messer, A., Greeberg, I., Bernadat, P., & Milojicic, D. (2002). *Towards a distributed platform for resource-constrained devices.* Proceedings of the IEEE 22nd International Conference on Distributed Computing Systems (ICDCS'2002), Vienna, Austria.

Milojicic, D., Messer, A., Bernadat, P., Greenberg, I., Fu, G., Spinczyk, O., et al. (2001). *Pervasive services infrastructure* (Tech. Rep. No. HPL-2001-87). HP Laboratories, Palo Alto, CA.

Ninja. (2001). The Ninja project. Retrieved August 15, 2004, from *http://ninja.cs.berkeley.edu*

Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., & Walker, K. R. (1997). *Agile application-aware adaptation or mobility.* Proceedings of the 16th ACM Symposium on Operating Systems Principles, France.

Wand, A. I., & Chunnian, L. (2001). Process support for mobile work across heterogeneous systems (Tech. Rep.). Norwegian University of Science and Technology, Department of Information Sciences.

Yunos, H. M., Gao, J. Z., & Shim, S. (2003, May). Wireless advertising's challenges and opportunities. *IEEE Computer*.layers: network, service discovery, service composition, service execution, and application.

Endnote

[1] The use of the location criterion is backed by the work of Chakraborty, Perich, Joshi, Finin, and Yesha (2002). In this work, a reactive service composition architecture for pervasive computing environments has been designed. The architecture consists of five layers: network, service discovery, service composition, service execution, and application. We focus on the service execution layer. During the execution of services, this layer might want to optimize the bandwidth required to transfer data over the wireless links between services and, hence, execute the services in an order that minimizes the bandwidth utilization. This optimization is similar to the location criterion. With that criterion, the cross-network traffic between the resources can be reduced, which avoids extra data exchanges between distant resources.

Chapter XII

Service-Oriented Computing Imperatives in Ad Hoc Wireless Settings

Rohan Sen
Washington University in St. Louis, USA

Radu Handorean
Washington University in St. Louis, USA

Gruia-Catalin Roman
Washington University in St. Louis, USA

Christopher Gill
Washington University in St. Louis, USA

Abstract

Service-oriented computing is the latest step in a progression of programming paradigms containing, among others, the object-oriented computing and component-oriented computing paradigms. The service-oriented computing paradigm is characterized by a minimalist philosophy, in that a user needs to carry only a small amount of code in its local storage, and exploits other services by discovering and using their capabilities to complete its assigned task. While the paradigm was born and reached a certain level of maturity in wired networks, we examine the imperatives for service-oriented computing in ad hoc wireless networks. An ad hoc wireless network is a dynamic environment by necessity, which exhibits transient interactions, decoupled computing, physical mobility

of hosts, and logical mobility of code. The motivation for this chapter is to understand the imperatives for a viable service-oriented computing framework in ad hoc wireless settings.

Introduction

Service-oriented computing is a new paradigm that is gaining popularity in distributed computing environments due to its emphasis on highly specialized, modular and platform agnostic code facilitating interoperability of systems. It borrows concepts from more mature paradigms, such as object-oriented and component-oriented computing. This results in a progression from object-oriented computing to component-oriented computing and, finally, to service-oriented computing, a new paradigm for designing and delivering software. Just as an object encapsulates state and behavior at a fine level of granularity, a service offers similar encapsulation at a larger scale. This evolution raises the level of abstraction at which systems are engineered, while preserving beneficial properties such as modularity, substitution, and encapsulation. Every participant in a service-oriented computing system is a provider or user of a service, or both. The service-oriented computing paradigm is characterized by a minimalist philosophy, in that a user needs to carry only a small amount of code in its local storage, and exploits other services by discovering and using their capabilities to complete its assigned task.

In this chapter, we examine the imperatives for service-oriented computing in ad hoc wireless networks. Ad hoc wireless networks are collections of hosts capable of wireless communication. Hosts within proximity of each other opportunistically form a network which changes due to host mobility. An ad hoc wireless network is a dynamic environment by necessity, which exhibits transient interactions, decoupled computing, physical mobility of hosts, and logical mobility of code. An important class of ad hoc mobile systems is based on small, portable devices, and this class of systems is the focus of this chapter. Such devices have limited storage capacity and battery power which restricts the number of programs they can store and run locally. Service-oriented computing offers a solution to this problem. By its very nature, service-oriented computing is designed to facilitate sharing of capabilities while minimizing the amount of functionality a single host needs to possess. Such a design is especially effective in ad hoc networks where storage space on individual hosts is at a premium, yet a large number of hosts can contribute small code fragments resulting in a rich set of capabilities being available in the network as a whole.

Service-oriented computing has received much attention from researchers worldwide. However, most of this work has been focused on architectures and implementations for wired networks.

Migrating service-oriented computing to ad hoc networks is nontrivial and requires a systematic rethinking of core concepts. Many lessons have been learned from the work done in the wired setting, especially regarding description and matching of services. However, the more demanding environment of an ad hoc wireless network requires novel approaches to advertising, discovering, and invoking services. We envision such ad hoc networks being used in a range of application domains, such as response coordination

by firefighters and police at disaster sites or command and control by the military in a battlefield. Such scenarios demand reliability despite the dynamic nature of the underlying network.

A key issue with service-oriented computing in ad hoc networks is to mitigate the problem of frequent disconnection and to ensure that some channel between the user and the provider of a service is maintained for a sufficient duration. To deal with this, we introduce a new concept that considers the issue of automatic and transparent discovery of services and the maintenance of channels between an application and the services needed to carry it out. We call this concept *context sensitive binding*, a novel way to maintain the service provision channel between two entities dynamically as they move through changing physical and logical contexts. This helps decouple concerns about network availability and connectivity from concerns specific to service-oriented computing. It also facilitates simplifications in the software development process. Context sensitive binding is put forth in this chapter as the foundation for reliable service advertisement, discovery, invocation, and composition in ad hoc networks. Among other things, this idea promises to facilitate the porting of wired network technologies to ad hoc wireless networks as it mitigates some of the problems inherent to ad hoc networks. We discuss these issues in more detail in the main section of this chapter.

The motivation for this chapter is to understand the imperatives for a viable service-oriented computing framework in ad hoc wireless settings. We begin by examining current technologies, algorithms, and capabilities that have been implemented for use in wired networks as a baseline. We then extend these concepts to cater to the special challenges of service-oriented computing in ad hoc networks and direct the reader's attention to research issues in this area. The rest of the chapter is organized as follows. We cover information on existing service-oriented computing architectures and the *Semantic Web* effort in the Background section. The section on Ad Hoc Wireless Network Perspective on Service-Oriented Computing represents the main thrust of this chapter and discusses the elements of a service-oriented computing framework, examining current technologies alongside our ideas on how these concepts may be applied to ad hoc networks. We cover potential areas of research in the Future Trends section. We summarize in the Conclusion section.

Background

In this section, we define the elements that make up a service-oriented computing framework and then review some of the existing models that have been proposed using these elements. Finally, we highlight the *Semantic Web* as an existing example of service-oriented computing.

Characterizing the Service Elements

A service-oriented computing framework is a conglomerate of elements, each element fulfilling a very specific role in the overall framework. We list the salient elements required

for a viable service-oriented computing framework and the criteria used to judge their quality. We use this list as a basis for future discussion.

- The *service description* element is responsible for describing a service in a comprehensive, unambiguous manner that is machine interpretable to facilitate automation and human readable to facilitate rapid formulation by users. The aim is to specify the functions and capabilities of a service declaratively using a known syntax. A good service description mechanism must have a clear syntax and well-defined semantics which facilitate matching of services on a semantic level.
- The *service advertisement* element is responsible for advertising a given service description on a directory service or directly to other hosts in the network. The effectiveness of an advertisement is measured as a combination of the extent of its outreach and the specificity of information it provides up front about a service, which can help a user determine whether he or she would like to exploit that service.
- The *service discovery* element is the keystone of a service-oriented computing framework and carries out three main functions. It *formulates* a request, which is a description of the needs of a user. This request is formatted in a similar manner to the service description. This element also provides a *matching function* that pairs requests to services with similar descriptions. Finally, it provides a mechanism for the user to *communicate* with the service provider. A good discovery mechanism provides a flexible matching algorithm that matches advertisements and requests based on their semantics and maximizes the number of positive matches giving the user a more eclectic choice of services for his or her needs.
- The *service invocation* element is responsible for facilitating the use of a service. Its functions include transmitting commands from the user to the service provider and receiving results. It is also responsible for maintaining the connection between the user and the provider for the duration of their interaction. A good invocation mechanism abstracts communication details from the user and, in the case of network failure, redirects requests to another provider or gracefully terminates.
- The *service composition* element provides mechanisms to merge two or more services into a single composite service which combines and enhances the functions of the services being composed. A good service composition mechanism leverages off each of the elements listed above to provide automatic composition of services without human intervention.

Using Service Elements to Build Models

The elements described previously form the building blocks for most service-oriented computing models. Various models entail the use of some or all of the elements described, depending on their complexity. We review some of the more popular models as background to our work and highlight features unique to each model. The Service Location Protocol (SLP) is designed for use on corporate LANs. Universal Description, Discovery and Integration (UDDI) and Universal Plug and Play (UPnP) are designed for use on the

Semantic Web. *Salutation* is a general service model originally created with communication between appliances and other equipment in mind. Jini is a more general model that can be used on the Web and with a wide range of applications.

The **Service Location Protocol** (SLP) (Kempf & Pierre, 1999) was developed by the SRVLOC working group of the Internet Engineering Task Force (IETF) with the idea of creating a service-oriented computing standard that was platform independent for the Internet community. In SLP, every entity is represented as an agent. There are three kinds of agents: User Agents which perform service discovery on behalf of clients, Service Agents which advertise locations and capabilities on the behalf of services and register them with a central directory, and Directory

Agents which accumulate service information received from numerous Service Agents in their repository and handle service requests from User Agents. An interesting feature of SLP is that it can operate without the presence of a Directory Agent, that is, it does not require a central service registry to function. User Agents search for a Directory Agent by default, but if they do not receive any replies, they continue to operate directly with peer agents. If a Directory Agent starts operating at some point in the future, it advertises its presence and the User and Service Agents that receive the advertisement automatically start using the Directory Agent as a central service repository.

SLP registers services using templates which follow a specific pattern. Requests are made using a similar template though the parameters differ slightly. SLP's ability to operate in the absence of a Directory Agent makes it especially useful in ad hoc environments where it is not feasible to have a central service registry. Overall, SLP offers a clean model that is easy to conceptualize, and, due to its design, it can be implemented readily in modern languages like C++ and Java.

Universal Description, Discovery and Integration (UDDI) (UDDI Organization, 2000) was formulated jointly by IBM Corporation, Ariba Incorporated, and Microsoft Corporation. UDDI technology is aimed at promoting interoperability among Web services. It specifies two frameworks, one to describe services and another to discover them. UDDI uses existing standards such as Extensible Markup Language (XML) (XML Core Working Group, 2000), Hypertext Transfer Protocol (HTTP) (Fielding et al., 1999), and Simple Object Access Protocol (SOAP) (XML Protocol Working Group, 2003).

The UDDI model envisions a central repository which is called the UDDI Business Registry (UBR). A simple XML document is used to specify the properties and capabilities of a service. Registration with the UBR is done using SOAP. Information present in a service description can include things like the name of the business that provides the service, contact information for people in charge of administering the service, and identifiers for the service and descriptions. The UBR acts as a mediator and assigns a unique identifier to each business and each service. Marketplaces, search engines, and enterprise-level applications query the UBR for services, which they use to integrate their software better with other business entities like suppliers, dealers, and so forth.

The UDDI model is distinctive in that its approach looks at service-oriented computing from a business process perspective. It envisions electronic interactions between businesses resulting in trading of services and goods, much like the business-to-business (B2B) model today but with enhanced capabilities and features. With the

backing of industry giants like Microsoft and IBM, the future of this technology looks promising.

Universal Plug and Play (UPnP) (Microsoft Corporation, 2000) was developed by Microsoft Corporation to facilitate seamless communication between networked devices in close proximity to each other. UPnP leverages TCP/IP and the Internet for its communication needs. It assumes a network that is dynamic with devices frequently joining or leaving. A device, on joining a network, conveys its own capabilities on request and learns about the capabilities of other devices. When it leaves the network, a device does so without leaving behind any explicit evidence that it was there. Entities in the network may be controllable devices or controllers. Controllers actively search for proximal devices. The network is *unmanaged* in that there is no DNS or DHCP capability. Instead, a mechanism called AutoIP (Miller, 2003) is used to allocate IP addresses.

In UPnP, a client, also called a *control point*, can undertake five kinds of actions. The discovery action is based on the Simple Service Discovery Protocol (SSDP) (Goland, Cai, Leach & Gu, 1998). It exchanges information about the device type, an identifier, and a URL for more detailed information. During the description action, the control point uses the URL obtained during the discovery action to get a detailed XML description of the device. Then the control point sends a control message encoded in XML to a Control URL using SOAP to discover the actions and parameters to which the discovered device responds. The event action consists of a series of events formatted in XML using the General Event Notification Architecture (GENA) (Cohen & Aggarwal, 1998) which reports changes in the state of the service. The presentation action consists of presenting a URL that can be retrieved by a control point, presumably in a format that allows an end user to control the discovered device. At this point, UPnP assumes some external architecture and protocol that handles subsequent interactions with the device.

UPnP uses multicast for discovery and unicast for service utilization and event notification. Services and controllers are written using an asynchronous, multithreaded application model.

UPnP requires a Web server to transmit device and control descriptions, though this server is not required to be on the device itself. This requirement means that UPnP works best on the Web and porting it to other environments, especially those with low-power hosts, is nontrivial.

Salutation (Salutation Consortium, 2003) is an open standard service discovery and utilization model formulated by the Salutation Consortium. The vision for Salutation is not on the scale of the World Wide Web. Instead, Salutation hopes to promote interoperability among heterogeneous devices in settings such as corporate LANs where there is permanent connectivity from either wired networks or wireless gateways, and disconnection is not an issue. In addition, Salutation strives to be platform, processor, and protocol agnostic.

Salutation has two major components:

- (1) the Salutation Manager (SLM) which presents a transport-independent API called the SLM-API and

- (2) The Transport Manager (TM) which is dependent on network transport and presents an SLM-TI interface between the Salutation Manager and the Transport Manager.

Services are registered via a local SLM or by a nearby SLM connected to a client. Service discovery occurs when SLMs find other SLMs and the services registered with them. Capability exchange is done by type and attribute matching. The SLM protocol uses Sun's ONC RPC (Srinivas, 1995). Periodic checks by the client ensure that it has the most current list of available services.

Salutation is interesting in that it solves a highly focused but widely relevant problem. An ideal environment for a Salutation deployment is a busy office space where there are a lot of computers, printers, fax machines, and other electronic devices. Using Salutation to automatically discover and use devices within range eliminates the effort of individually configuring every single device. This makes Salutation a useful productivity-enhancing tool.

Jini (Edwards, 1999) is a distributed service-oriented model developed by Sun Microsystems. Services in a Jini system can be hardware devices, software components, or a combination of the two. A Jini system has three types of entities: service providers, lookup services, and users. A service provider multicasts a request for a lookup service or directly requests a remote lookup service known to it *a priori*. Once the handle to a lookup service has been obtained, the service provider registers a proxy object and its attributes with the lookup service. The proxy object serves as a client-side handle to the actual service. The user makes a request for some service by specifying a Java type (or interface that the desired service must implement) and desired attributes. Matching is done based on type and values. Once a candidate service is found, the proxy object registered by the service provider is copied to the client using RMI (Sun Microsystems, 2003b). Clients use the proxy object to interact with the service.

The Jini model is designed for ubiquitous computing and is intrinsically scalable. It is language and protocol independent (though Java and TCP/IP seem to be natural choices for an implementation) and is resilient because multiple lookup services ensure that there is no single point of failure. Jini holds much promise for middleware developers that use the Java programming language and has been formulated with Java-like languages in mind. Also, Jini is unique in that it introduces the idea of shipping a proxy object to the client where it is used as a handle to the actual service. The proxy approach allows complex services to reside on a server at a well-known location. The service then simply has to encode its well-known location within its proxy and can then be called from any client without that client having any knowledge of the actual location of the service. It also mitigates the issue of establishing the protocol between the service and the user since the proxy object hides from the end user all such communication.

These five models illustrate how various elements must come together and work as a cohesive whole to deliver the promise of service-oriented computing. We now highlight the practical feasibility of such models by focusing on a model that promises to evolve the World Wide Web into a service-oriented *Semantic Web*. While much work has already been done on this model, efforts continue to refine and augment its core.

A Service-Oriented Computing Model at Work: The Semantic Web

The largest deployment of a service-oriented computing model is the *Semantic Web* (Berners-Lee, Hendler & Lassila, 2001). The Semantic Web effort aims to add logic to the current World Wide Web so that machines can infer the meaning and spirit of the content they handle. The vision is to evolve the Web into a collection of entities, each of which may be a user or provider of services. The key technologies needed to deliver the *Semantic Web* concept are (1) innovative naming systems that provide simple constructs to describe complex objects, (2) relations that are machine interpretable, and (3) flexible matching algorithms that can match user requests with services based on the meaning of the request and service advertisement. Some issues of service-oriented computing, such as providing a central service registry and reliable links between user and provider, are mitigated since we can leverage the existing infrastructure of the World Wide Web. Hence, the discovery and invocation elements are not especially significant in the Web setting.

The *Semantic Web* uses a layered naming system. Lower level languages such as the Web Services Description Language (WSDL) (W3C XML Activity on XML Protocols, 2003) describe *how* the data is sent across the wires. This layer handles all application layer protocol issues. Higher level languages describe *what* is being sent across and *why*. High level description languages are required to be clear, concise, and support easy matching between two entities. Ontologies are high level languages that capture the semantics of an entity and its relation to other entities. An ontology is often itself structured into layers. DAML+OIL (Horrocks, 2002) is a combination of a markup language to construct ontologies (DAML) and an ontology inference layer (OIL) to interpret the semantics of the description. It is currently the language of choice for formulating ontologies for Web services. The lowest layer of the system provides the syntax and is encoded in XML (XML Core Working Group, 2000) since it is a W3C standard and, hence, has maximum outreach. Above the syntax layer is a framework that provides a basic set of objects and constructs to describe entities and relations. In DAML+OIL, the Resource Description Framework (RDF) (W3C Semantic Web Activity, 2003), also a W3C standard, fulfils this functionality. However, RDF is not powerful enough to describe entire ontologies. DAML+OIL fills this gap by extending the RDF concept to provide more constructs and relations. Finally, DAML-S (Ankolekar et al., 2002) is an ontology specific to Web services that has been built using DAML+OIL.

DAML-S can be used to encode both service advertisements and service requests. The only remaining element is a matching algorithm that matches a service advertisement to a request. Conventional matching algorithms perform exact matches, which are considered too rigid for this purpose. Many inexact matching algorithms have been proposed to date, but due to constraints of space, we discuss just one suggested by Paolucci, Kawmura, Payne, and Sycara (2002) for the *Semantic Web*. The algorithm compares requests and advertisements and assigns a degree of similarity to each pair. Valid degrees of similarity are: *exact, plugIn, subsumes,* and *fail* with *exact* being the strongest and *fail* being the weakest in the hierarchy. For every request, the algorithm returns the request-

advertisement pair that has the strongest degree of matching. This allows inexact matches to be returned, which improves the flexibility of the system.

The *Semantic Web*, though still in its infancy, is a good illustration of the potential for service-oriented computing in wired networks. In the next section, we explore the potential of service-oriented computing in ad hoc wireless networks and discuss the imperatives, issues, technical challenges, and benefits of migration to such a new environment.

The Ad Hoc Wireless Network Perspective on Service-Oriented Computing

In the previous section, we described the salient elements of a service-oriented computing framework and reviewed some existing models. In this section, we revisit the same elements but from an ad hoc wireless network perspective. We discuss issues relevant to ad hoc networks and offer solutions in certain cases. From our discussion, we distill imperatives for service-oriented computing in ad hoc wireless settings. To facilitate our discussion, we make use of an example and highlight how each element plays a unique and key role in the illustrated interaction.

Example: A Discerning Tourist

David is a researcher working in St. Louis, USA. He plans to attend a conference in London, UK, to present a paper. However, due to his busy schedule in London, he only has one day for sightseeing. David is a history buff so he is only interested in visiting places of interest that have historical significance. However, David is not familiar with London since this is his first visit. Hence, he needs some help planning his day of sightseeing.

On the flight to London, David enters a request on his PDA for tourist information services and indicates his preference for historical sites. On his arrival in London, David waits for his baggage in the arrivals lounge, but his PDA immediately starts working. It begins communicating with wireless information kiosks in the arrivals lounge. After a few seconds of searching, it finds a London Tourism Bureau service which lists all tourist attractions around London. David's PDA automatically transmits David's pre-entered preferences to the service and gets a modified list containing only historical sites. David sees the list and checks off the sites in which he is most interested. At the same time, he requests directions to each of these sites. The London Tourism Bureau service has no map data to provide directions. Hence, the service automatically queries the London Map service to get directions to each attraction. David now notices that all these attractions are far away from his hotel, and since he does not have access to a car, requests an itinerary that uses only public transport. The tourism service now queries the London Transport service and gets times of buses and trains and compiles a full itinerary for

David. As a last step, it queries the BBC Weather service for a weather forecast and then rearranges the itinerary so that outdoor sites are scheduled when there is a lesser chance of inclement weather. The compiled itinerary is then transmitted back to David, who saves it on his PDA. Using a complex set of services, David has now managed to plan his day of sightseeing in London while waiting for his bags. In the following subsections, we show how each element fulfills a portion of the functionality required to deliver such services to users.

Description

The description element is responsible for dealing with issues related to creating a *profile*, a comprehensive and unambiguous description of a service. A comprehensive description of a service ensures the service can be used to the maximum extent of its capabilities. The unambiguous characteristic of the service description will ensure consistent interpretation of the data in the profile. The profile must be machine readable to facilitate the automation of the search process.

To make a profile machine readable, a well defined syntax must be imposed on the service description. Some service discovery infrastructures use simple data structures to describe their service profiles, but most use markup languages like the Resource Description Framework (RDF) (W3C Semantic Web Activity, 2003) and DAML (Horrocks, 2002). XML (XML Core Working Group, 2000) is used for syntactic purposes. In some cases, the service profile can be replaced with an introspective analysis of the code. Using a technique called reflection, the code of a service can be inspected and its capabilities inferred from the list of methods it contains and their parameters. This technique is useful in languages that have reflection built in, as this mitigates the overhead of formulating a service profile.

Once a common syntax for the description has been established, the information in a service description can be broken into tokens and automatically processed by algorithms. However, the tokens have no value unless they have semantics associated with them. Semantics can be defined by introducing a type system and establishing relationships between entities. RDF (W3C Semantic Web Activity, 2003) describes relations between entities using directed graph-like data structures. Its Resource Description Framework Schemas (RDFS) (W3C Metadata Activity, 2000) extension provides additional basic notions such as classes. Details of the protocol to communicate with the service, if required, can be described using languages like WSDL (W3C XML Activity on XML Protocols, 2003).

Syntax and semantics are combined in ontologies. Ontologies define specific terms for certain contexts and encode relationships between elements of a service description that help in matching the client's request to the right service profile. For example, the London Tourism Bureau ontology may have the HistoricalSite class be a child of the GenericTouristSite class, but the London Map ontology may not even contain the GenericTouristSite class. The parties involved in the transaction need to make sure they understand the same semantics from the information represented in the service profile; that is, they need to share a common ontology.

There is an ongoing effort to develop ontologies for specific fields of activity building up a global scale standardization system. The reader is pointed to http://www.daml.org/ontologies/ for samples of such ontologies.

The mechanics of describing a service essentially do not change when we move to ad hoc networks. Certainly, the syntax can be reused and the semantics remain the same. However, it would be remiss to say that a service profile tailored for use in a wired network would work as effectively in an ad hoc environment. This is because the ad hoc network is a more demanding and dynamic environment. The service profile has to be enhanced to meet these changes. Examples of enhancements include adding location information, motion profiles (if the service provider is mobile), possible alternate providers in case a provider moves out of communication range, battery power remaining, trust certificates, and other such details.

Distilled Imperatives for Description in Ad Hoc Settings

- The service profile must be comprehensive, unambiguous, and machine interpretable.
- A common ontology is required to ensure that both user and provider infer the same semantics for an entity.
- The profile should be enhanced with details pertaining to the mobility and available resources of the service provider.

Advertisement

In our example, David's PDA automatically discovered the tourism service when he arrived in London. This was a two-way process by which David's PDA requested such a service, and the service advertised itself. In this subsection, we discuss issues pertaining to advertising a service. Simply put, advertisement is the process of putting the service profile and its ancillaries in a place where it can be read by anyone so that users are made aware of the services' existence and can use them when they need to.

Advertisements come in many different flavors, ranging from simple text to complex objects. The simplest kind of advertisement is a plain text description of a service, for example, encoded in XML (XML Core Working Group, 2000). This kind of advertisement is used widely on the *Semantic Web*. Next in order of complexity is the two-step form of advertisement used by the UPnP (Microsoft Corporation, 2000) system. The first step conveys information about the existence of the service and offers a Uniform Resource Identifier (URI) for a more comprehensive description. On obtaining this URI, the user connects to the specified location to get a complete description. The most advanced kind of advertisement is the proxy object approach used by the Jini (Edwards, 1999) system. The proxy object is advertised along with the service profile text. The proxy object behaves like a client-side handle to the provider of the service and is used to control interactions between the user and provider.

Services may advertise themselves in two ways: on a central service registry or directly to users in a peer-to-peer fashion. The central registry is easier to maintain and configure. However, this design may introduce single points of failure. In such a setting, the complete failure of a lookup service could paralyze an entire community of hosts trying to interact. Another issue inherent in central directories is services that become suddenly unavailable (for example, due to their host crashing or disconnecting) but which leave behind their advertisements in the registry. Lease agreements for services can be used to mitigate this issue but do not eliminate it. Finally, using a central registry means that the knowledge of the location of this registry may need to be disseminated using other channels. Using our example, David would have to know *a priori* the address of the London service registry and configure his PDA before he could discover the tourism service. While this is not unreasonable, it does detract from the plug-and-play characteristic of the system.

The second approach to advertisement is the registry-less technique used by SSDP (Goland et al., 1998). Clients and servers advertise their needs and/or presence directly using unicast (when the location of the party is known) or multicast. The announcements happen when the process comes up and possibly periodically thereafter. The advantage of this approach is that it eliminates the need for a third-party mediator. Orphan advertisements cannot exist due to the simple nature of the mechanism. The downside of the approach is the higher bandwidth usage and the limited scope of users an advertisement can reach. In the context of our example, the tourism service would advertise itself individually to David's PDA and everyone else in proximity, whether they asked for such a service or not. Hybrid approaches exist that use both service directories and direct announcements. Users would then try to contact a service registry manager process, and if it were not available, they would start interacting in a peer-to-peer fashion.

An effective advertisement mechanism for the ad hoc setting needs to consider two main issues. The first is the size of the advertisement. Advertisements need to be as small as possible so that they do not monopolize bandwidth and can be transmitted in the short periods of time in which connectivity may exist. The second is the manner of advertising. Services should be advertised in a transiently shared space that adapts to host mobility. Neither of the approaches discussed, for example, central registry or peer-to-peer are suitable. By nature of the network, a central well-known registry is not feasible and direct peer-to-peer advertisements use up a lot of bandwidth and battery power on resource poor hosts.

The solution to the problem lies in the use of global virtual data structures. Lime (Murphy, Picco & Roman, 2001) is a Java implementation of the Linda (Gelernter, 1985) coordination model, adapted for ad hoc networking. The mechanism used for advertising services is based on transient sharing of a local service registry (Handorean & Roman, 2002; Storey, Blair & Friday, 2002). The content of the service registry is distributed across participating hosts and processes. Each process has its own local registry which it shares with local registries held by other processes, forming a federated service registry. The content of the federated registry is atomically updated as processes join or leave the community. When a process leaves the community, the advertisements in its local registry become inaccessible but the remaining processes are not affected. The approach guarantees the consistency of the service registry content at all times, in that a service cannot be

discovered if it is not available. Returning to our example, the tourism service would put its service advertisement in its local registry. When David landed in London, his PDA would merge its local registry with the tourism registry and exchange data. Once the data transfer was completed, the two registries could separate.

Distilled Imperatives for Advertisement in Ad Hoc Settings

- The size in bytes of the advertisement should be minimized.
- Advertisements should be placed in a transiently shared space that is accessible by all interacting parties.
- The advertisements in the registry should be consistent and there should be no advertisements without there being an associated active service.

Discovery

The process by which a user finds a service provider is called service discovery. Service providers do the best they can to get their advertisements out to interested users. However, as mentioned in the previous subsection, there is essentially a two-part process of (1) advertising by the provider and, (2) requesting by the user. Without the user's request, there is no relation established between the user and provider. Requests are also referred to as templates that describe the required characteristics of a service.

Like advertisements, requests also have varying levels of complexity. The simplest request contains the identifier of a service the client already knows. This scenario assumes that the client knows the protocol and the semantics of interacting with the provider and only needs to discover the provider's presence. When the client runs for the first time in a new environment or needs to search for a service it has never used before, the request becomes more detailed. Such a request is formulated as a template that describes the capabilities that the service must implement and performance parameters in the form of attribute-value pairs. The formulation of a request follows the same guidelines that apply to the description of a service. A more advanced request breaks up the required list of attributes into *must have* and *can have* attributes. A candidate service is rejected if it does not have any of the *must have* attributes while the *can have* attributes behave like bonuses and are used as tie-breakers by the matching algorithm.

Like a service advertisement, a service request can be broadcast or multicast as repetitive or periodic messages. Indirect requests unicast their messages to a central service registry and obtain results from there. Models, such as Salutation (Salutation Consortium, 2003), support service brokers which are processes that manage service registries. The servers contact these brokers to advertise their services while the clients contact them to address their requests for services. If the model is based on service repositories and service brokers (lookup services), both clients and servers need to discover these special services first.

The most critical piece of the discovery mechanism is the algorithm that matches requests to advertisements. Matching algorithms themselves constitute a comprehensive research area: due to limitations of space, we mention them only briefly here. Matching can take two forms. *Exact matching* is the type of matching that is most common today and represents a logical equivalence operator. The other form is inexact matching, which seeks to discover essential commonality of features of the two entities being matched. Even when two entities are not equivalent, a positive match is returned if the differences do not matter. This is a nontrivial process since teaching a machine to make correct judgment calls is a very complex task.

An exact matching policy is, in many circumstances, considered too rigid for the purpose of matching services. Consider our example where David's PDA made a request for "sightseeing services" whereas the advertisement was for a "tourism service". An exact matching algorithm would not return a match, but, in fact, the two are a good match for each other, even though they do not use the same terms. A first step towards a more general matching policy is to allow for polymorphic matching, for example, a client looking for a transportation service could be given a handle to a bus service if a bus service is, in fact, a subclass of a transportation service. Another type of inexact matching is inexact value matching or approximation; for example, David may be looking for a taxi to take him from the airport to the hotel, but he would accept a limousine if no taxis were available. The client can specify a range or set of values for a certain attribute rather than a single rigid value. It is also important for the client to be able to specify what the best match looks like in these cases. In some cases, the lower value is better while, in other cases, the higher value is better. The best choice is the result of optimizing a function over the range or set of values allowed by the client. *Semantic matching* as discussed in Paolucci et al. (2002) is an inexact method of matching services based on their semantics and holds much promise.

In an ad hoc networking environment, the clients must be more flexible in requesting and using services. Allowing for flexible matching is extremely important in an environment where all interactions are opportunistic and many of the participants may be resource poor. While an inexact matching policy seems very important in ad hoc networks, it comes at a cost. Exact matching can return the first match and terminate. Inexact matching needs not only to verify the relationship between the advertised value and the acceptable range, but it also needs to compare matches to all other available matches. Simply discovering a match does not mean that another advertisement, not yet considered, could not yield a better match. These enhancements to the requests formulated by clients and to the matching algorithms may lead to costly searches. In ad hoc networks, this can be critical. While in a wired setting a client can afford to browse a service repository, in ad hoc settings the connectivity may not be guaranteed for such a long period of time. This constraint affects the type of search a client performs. A synchronous search usually assumes reliable connectivity, but for an ad hoc networking environment, an asynchronous approach can be better for several reasons. The client delegates the effort to perform the search to another party (possibly a searching service) and, if possible, continues execution while waiting for a notification. This approach abstracts issues of consistency of the data in the service registry away from the client. The result of a match is usually a handle to a service. Returning multiple handles forces the client to filter them locally and to choose the best fit. In ad hoc networks with low power hosts, this can become a

problem as discovering multiple services and filtering them can be resource intensive. Hence, matching algorithms designed for ad hoc networks must be designed so that the filtering occurs on the provider side rather than the client side.

When a service handle is returned, it is paired with a service identifier. This can be used by the client to access the service in the future, provided it is within communication range. The service provisioning framework ensures that the ID is globally unique. Besides the service ID, the client can receive the provider's profile, a URI to the provider profile, or a proxy object that represents the provider locally to the client. The simpler the return type, the more complex the processing on the client side. This requires more sophisticated clients which are able to filter the results themselves, learn communication protocols, and so forth. The advantage is reduced bandwidth consumption and framework simplification at the expense of more complex clients. An alternate solution is the proxy object approach. At the expense of higher bandwidth usage, the client retrieves a proxy object that abstracts details of communication with the provider. When it is no longer needed, the proxy object can be discarded, thus, saving memory space. The proxy idea is implemented in Jini (Edwards, 1999). Jini uses RMI (Sun Microsystems, 2003b) to download the proxy object locally to the client and then to mediate the interaction between the proxy and its server, even though they may communicate without using (Sun Microsystems, 2003b). The proxy object approach is more elegant, but it brings its own challenges; for example, it implies a mechanism for code mobility as suggested in ìCode (Picco, 1998). From the ad hoc perspective, the trade-off between bandwidth and power is a challenge as both commodities are in short supply in the ad hoc setting. In general, solutions differ appropriately from system to system.

Distilled Imperatives for Discovery in Ad Hoc Settings

- The request should be comprehensive to enable semantic matching which can maximize hits and is useful in environments with transient interactions and decoupled computing.
- The matching algorithm should be flexible and should return the fewest possible results to reduce client-side processing.
- The handle to the service should have a small footprint but should be able to abstract details of communication and coordination in ad hoc networks away from the user of the service.

Invocation

The next step after discovering a service is to use or invoke it. This step is one of the most challenging in the ad hoc wireless setting. Consider the scenario in our example where David is downloading his itinerary from the tourism service. When the download is half completed, David unknowingly walks out of range of the transmitter. This breaks the

connection and terminates the download. This scenario is typical of an ad hoc network where frequent and unannounced disconnections abound.

The biggest challenge is to maintain connectivity for a sufficiently long period of time. If we could assume that the connectivity lasts for the entire duration of the service discovery and invocation, the dynamic character of an ad hoc network would be reduced to that of a reliable wired network. A novel solution which mitigates this problem is the notion of context-sensitive bindings. Context sensitive bindings create a binding between the user and provider of the service and try to maintain it for the duration of some interaction between them. If the provider goes out of range, the framework tries automatically to redirect the user to another similar provider or else to fail gracefully. Context-sensitive bindings can also use the notion of a *motion profile* published by a service provider in its advertisement to estimate disconnections before they occur and to judge how long the disconnection will last. If an imminent disconnection seems to be permanent, a service invocation may be cancelled. However, if the analysis of the motion profile indicates possible future contact between the two parties, an asynchronous type of interaction can be employed. The client only needs connectivity initially until the method execution is launched, that is, until the parameters of the call are shipped, and the call is made on the remote host. The connection between the two ends can be dropped after that, until some result is ready for the client. Some notification mechanism must tell the client that there is a result ready or the client will need to poll for the result when the connection is restored. A more sophisticated version of the above example can entail result delivery to a remote location. It is possible to deduce from motion profiles that the client moves towards a certain destination while its task is being processed. Geocast (Ko & Vaydia, 1998) protocols can help deliver the result in a certain physical area while Mobicast (Huang, Lu & Roman, 2003) can help coordinate both the time and the place of result delivery.

Once the provision for a connection is made, the next consideration is the protocol that is to be used. The description, advertisement, and discovery styles employed by a service provision model have direct implications for the mechanism used for service invocation. The simplest discovery mechanisms use well-known remote procedure call protocols like RMI (Sun Microsystems, 2003b) or SOAP (XML Protocol Working Group, 2003). At the next level lie services that can *learn* the communication protocol described in the advertised service profile. The clients need to share an ontology with the provider offering the services to be sure they understand the description of the protocol. WSDL (W3C XML Activity on XML Protocols, 2003) is an example of an XML-based language for describing services and how to access them. The last category consists of models in which clients do not need to know anything about the communication protocol. They use proxy objects as local representations of services. These proxy objects may be sophisticated enough to implement the entire service themselves, or they may be client-side handles to the server providing the service. Each proxy is used by the client to interact with the actual implementation of the service. The proxy handles the communication with the provider, and the client only needs to have knowledge of the interface the proxy offers. We observe again that the client and the provider's proxy need to adhere to the same ontology. The service provider needs to address issues like the communication protocol, how to handle disconnections, and dynamic rebinding of the proxy to another instance of the service running on a different machine while the framework offers

support for downloading code and possibly introspection. This model of interaction is implemented in Jini (Edwards, 1999). Support for proxy downloading and remote method invocation is ensured by RMI (Sun Microsystems, 2003b), while the client is expected to know how to interact with the proxy object (for example, it may use a given Java interface which the proxy object implements).

Distilled Imperatives for Invocation in Ad Hoc Settings

- Provide a mechanism that mitigates the problem of frequent and unannounced disconnections.
- Use motion profiles to predict where a certain host will be at a given time and plan interactions using this information.
- Have a sophisticated proxy that can hide communication protocol details from the user and perform tasks like rebinding to better service providers without user intervention.

Composition

Service composition is the notion of taking two or more autonomous services and combining them to behave like one composite service. We illustrate the usefulness of this concept by referring back to our original example. Note that David initially requested a service which gave him a list of tourist attractions. However, as he started receiving information from the service, he realized that he would require maps and directions. The tourism service did not have map data so it went to a map service and a weather service to find relevant data and combined it with the list of tourist attractions. In other words, the tourism service *composed* itself with the map and weather services.

Service composition is a powerful concept that allows multiple stand-alone services, each of which is highly specialized to a particular task, to be combined into a single service which can then offer an interlocked collection of services under a single umbrella. The benefit of the composed service is generally greater than the sum of benefits yielded by its constituents.

Composition promotes specialization in the development of services. In our example, if the tourism service had to develop its own mapping program, it would most likely be inferior to that of a dedicated mapping service. By composing itself with a third-party mapping service, it effectively *outsources* the problem of mapping the route, saving the cost of development of a mapping tool while, at the same time, providing a better quality of service. Composed services can differ based on the way they are constructed and used. We first discuss ways of building a composed service, followed by ways a composed service may be used.

The two main approaches for composing services are the distributed approach and the integrated approach. In the distributed approach, the composed service behaves like a distributed application with component services residing on different machines. The

composing mechanism may use the Façade design pattern (Gamma, Helm, Johnson & Vlissides, 1994) to provide a single common interface to the user (Mennie & Pagurek, 2000). The user makes local method calls to the interface which determines which of the component services can service the request and uses a protocol such as RMI (Sun Microsystems, 2003b) to call that service. Results are returned to the interface which disburses them to the user locally. In the integrated approach, the code for all the services is aggregated by the composing entity and runs locally. Regardless of the approach for composing a service, it is critical to ensure that the elements of the composed services neither conflict with each other nor cause the composed service to deadlock. In essence, if we have no services that have been successfully composed, adding the n+1st service should not violate the integrity of the n original services (Kirner, 2002). This can be verified in some cases through formal reasoning.

Composed services can be used in two ways. In automatic composition, the user makes some request to a specific service; for example, David requests a list of historical sites and the route that covers all of them from the tourism service. The tourism service checks its data and discovers that it can service the request for the list of sites, but it cannot provide route information. Hence, to fulfil the request, it automatically requests that a map service compute a route on its behalf. In object-oriented terms, this process is analogous to a method call on an external object. Notice that to David, all the data seems to be coming from a single source, the tourism service. In manual composition, the user manually selects a list of services that he or she believes will yield a useful result when combined. These services are then integrated according to the specific request. They do not seek out additional services to help them complete their tasks if they are lacking in some capability. The onus is on the user to ensure that such a situation does not arise.

We now discuss the merits of each approach as it pertains to ad hoc networks. The distributed approach to composing services is less reliable in ad hoc networks because a single element moving out of range can affect the entire composed service whereas if the services are hoarded, the connectivity needs to be preserved for a shorter period of time during which the services are copied over to the composing host. However, the downside of hoarding services is that everything needs to be ran locally which can drain power or tie up crucial memory. The best compromise in such a situation is to exploit the notion of motion profiles as discussed in the previous subsection and hoard only those services which are likely to move out of range and use the stable proximal ones in a distributed fashion.

For composed services, the automatic composition approach is more user friendly, but it does raise a concern that affects all composed services, that of security and confidentiality. Issues of security and privacy come to the fore in automatic service composition, especially if there is an exchange of personal information; for example, David might be willing to give his name and address to the tourism service since he knows it is reliable and has appropriate security measures to safeguard his information. However, he may not want his address transmitted to the mapping service for the purposes of computing the route since he does not trust their security systems. Such an event could lead to loss of privacy without the individual being aware of it. With manual composition, one can presume that the user would select only those services that he or she trusted.

Service composition is critical in the ad hoc wireless setting for two reasons. An ad hoc network by definition is an "anytime, anywhere" network. This means that an ad hoc network encounters a dynamic environment with its own nuances, requiring customized situation-specific services. Having large indivisible services which cater to specific needs is not flexible enough for such dynamic environments. It is much more effective to have a set of smaller services and select a subset of them to form a customized service on demand. Composition supports such a capability. The second reason why composition is an essential feature of ad hoc networks is that it allows for highly specialized services with potentially smaller footprints. This means that such services can run on small devices like PDAs or cell phones, increasing the number of devices in the network that can potentially be service providers and thereby increasing the number of basic and composed services available in the network.

The final issue we discuss is related to the control of a composed service with an eye toward security issues. Note that the elements of a composed service are provided by many different providers. Also observe that to use a provided service in a composition, the composing entity has to obtain some degree of control over the provided service. The provider may not necessarily want to relinquish control over its service and hence a middle ground must be found. We envision the use of management software, like Java Management Extension (Sun Microsystems, 2003a), which can manage services from different providers and yet allow them to retain some degree of autonomy.

DistilledImperatives for Composition in Ad Hoc Settings

- The approach to composing services should consider a hybrid strategy of integrated and distributed composition, which exploits the state of the network and services to the fullest.
- Composable services should be designed so as not to conflict with other services to increase the range of allowable compositions.
- Security and management systems should be compatible to maximize interoperability.

Future Trends

While service-oriented computing has reached a certain level of maturity in wired networks, it remains a nascent area of research in ad hoc wireless networks. Much potential for future research exists. In this section, we briefly describe some of the key research issues which we feel hold technical challenges and/or promise social impact. Service descriptions can be enhanced to include motion profiles, battery power levels, and composability. The content of a description could also be made dynamic to incorporate the *current* status of a service; for example, a printer would advertise how much paper is remaining in its trays. In the area of service advertisements, one can envision active advertisements where services do not wait for a request but actively solicit jobs. The idea of a client advertising jobs rather than providers advertising

services is also one that could be explored. Providers would examine a pool of jobs and pick up ones that they could perform. A provider would send a notification to the client when the job was completed. Another novel idea is time-limited advertisements, analogous to limited-time discount offers. A provider could offer lower cost or higher quality services at certain times when the load on the network is low, and the service is not being fully exploited.

In the arena of discovery, approximate matching based on semantics is an open area of investigation. These algorithms could be tailored to ad hoc networks with decision making based on the available content and urgency of requests. This direction of study could extend work on metrics for service discovery and usage, functions that help a user decide which is the best of two or more given services. Service invocation also poses some interesting problems; for example, the notion of follow-me services is one in which the client holds the service proxy, but the providing host changes over time due to physical mobility. This work can be related to resource leasing, pushing services onto resource-rich hosts and controlling them through proxy objects and delivering results remotely; for example, a client requests a job in St. Louis, shuts down, restarts in New York, and receives the result from a local access point. The concept of context-sensitive binding can be applied as well to facilitate active resource monitoring and load balancing based on network traffic.

The concept of composing multiple instances of a service for parallel processing for performance gains is extremely useful in ad hoc networks when the connection may exist for short periods of time only. This would require specification of dependencies between services with a hierarchy ranging from required to preferred. It also raises issues of combining code from various sources seamlessly and ensuring that the result exhibits correct and expected behavior.

Other than work related to enhancing the *elements* of service-oriented computing, there are some broader trends that can improve service-oriented computing as a whole. One such idea is the use of virtual currency, a version of which has been proposed in Buttyan and Hubaux (2001). Another concept is that of a language-independent representation of services, where the skeleton of the description highlights the semantics of the service and appropriate words from any language can be used to describe those semantics. This would allow the framework to understand requests in any language as long as the semantics of the service remained consistent. Lastly, one cannot ignore the security implications of such a framework which will require much research into online security, especially in scenarios where two programs can interact with each other without human intervention.

Conclusion

We began this chapter by introducing the notion of service-oriented computing and its applicability in ad hoc wireless networks. We discussed existing service-oriented computing models and highlighted the *Semantic Web* as an example of service-oriented computing at work. We then discussed issues of service-oriented computing in ad hoc wireless networks as they pertain to each of the salient elements of a service-oriented

computing framework, viz. description, advertisement, discovery, invocation, and composition. For each of these elements, we distilled a set of imperatives for the ad hoc wireless setting. Finally, we highlighted some key research issues and enhancements which we see as defining the next steps for research in this area. We hope this chapter has given a perspective on the issues, challenges, and imperatives of designing and implementing a service-oriented computing framework for ad hoc wireless settings, one that will serve as a guide for future research.

References

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., et al. (2002). *DAML-S: Web service description for the semantic Web.* Proceedings of the 1st International Semantic Web Conference.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). *The Semantic Web.* Scientific American.

Buttyan, L., & Hubaux, J. (2001). *Nuglets: A virtual currency to stimulate cooperation in self organized ad hoc networks* (Tech. Rep.). EPFL.

Cohen, J., & Aggarwal, S. (1998, July). General event notification architecture. Retrieved August 15, 2004, from http://www.globecom.net/ietf/draft/draft-cohen-gena-p-base-01.html

Edwards, W. K. (1999). *Core Jini*. Sun Microsystems Press.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999, June). Request for comments 2616 - Hypertext Transfer Protocol - HTTP/1.1. Retrieved August 15, 2004, from *ftp://ftp.isi.edu/innotes/rfc2616.txt*

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison Wesley.

Gelernter, D. (1985, January). Generative communication in Linda. *ACM Computing Surveys, 7,* 80-112.

Goland, Y., Cai, T., Leach, P., & Gu, Y. (1998, April). Simple service discovery protocol. Retrieved August 15, 2004, from *http://www.upnp.org/download/draft caissdp v1 03.txt*

Handorean, R., & Roman, G.-C. (2002, April). *Service provision in ad hoc networks.* Proceedings of the 5th International Conference on Coordination Models (pp. 207-219).

Horrocks, I. (2002). DAML+OIL: A description logic for the semantic Web. *IEEE Bulletin of the Technical Committee on Data Engineering.*

Huang, Q., Lu, C., & Roman, G.-C. (2003, April). Mobicast: Just-in-time multicast for sensornetworks under spatiotemporal constraints. In *Lecture Notes in Computer Science.* Springer-Verlag.

Kempf, J., & Pierre, P. S. (1999). *Service location protocol for enterprise networks: Implementing and deploying a dynamic service resource finder.* John Wiley & Sons.

Kirner, R. (2002, October). *Enforcing composability for ubiquitous computing systems*. Proceedings of the 7th Cabernet Radicals Workshop.

Ko, Y., & Vaidya, N. (1998). *Geocasting in mobile ad hoc networks.* Location-based Multicast Algorithms.

Mennie, D., & Pagurek, B. (2000, June). *An architecture to support dynamic composition of service components.* Proceedings of the 5th International Workshop on Component-Oriented Programming (WCOP 2000).

Microsoft Corporation. (2000, June). Universal plug and play device architecture. Retrieved August 15, 2004, from *http://www.upnp.org/download/UPnPDA10 20000613.htm*

Miller, S. (2003, October). The AutoIP publisher page. Retrieved August 15, 2004, from *http://www.autoip.net*

Murphy, A., Picco, G., & Roman, G.-C. (2001, April). *Lime: A middleware for physical and logical mobility.* Proceedings of the 21st International Conference on Distributed Computing Systems (pp. 524–533).

Paolucci, M., Kawmura, T., Payne, T., & Sycara, K. (2002). *Semantic matching of Web services capabilities*. Proceedings of the 1st International Semantic Web Conference.

Picco, G.-P. (1998, September). Code: A lightweight and flexible mobile code toolkit. *Lecture Notes on Computer Science, 1477,* 160-171.

Salutation Consortium. (2003, October). The Salutation consortium homepage. Retrieved August 15, 2004, from *http://www.salutation.org*

Srinivas, J. (1995, August). RFC 1831 - Open Network Computing Remote Procedure Call Protocol Specification. Retrieved August 15, 2004, *from http://www.ietf.org/rfc/ rfc1831.txt*

Storey, M., Blair, G., & Friday, A. (2002). MARE - Resource discovery and configuration in ad hoc networks. *Mobile Networks and Applications, 7,* 277-287.

Sun Microsystems. (2003a, October). Java management extensions homepage. Retrieved August 15, 2004, from *http://java.sun.com/products/JavaManagement/*

Sun Microsystems. (2003b, October). Java remote method invocation page. Retrieved August 15, 2004, from *http://java.sun.com/products/jdk/rmi/*

UDDI Organization. (2000). UDDI technical white paper. Retrieved August 15, 2004, from *http://www.uddi.org/pubs/*

W3C Metadata Activity. (2000, March). Resource description framework schema specification 1.0. Retrieved August 15, 2004, from *http://www.w3.org/TR/2000/CR-rdf-schema-20000327/*

W3C Semantic Web Activity. (2003, October). Worldwide Web Consortium page on resource description framework. Retrieved August 15, 2004, from *http:// www.w3.org/RDF/*

W3C XML Activity on XML Protocols. (2003, October). W3C recommendation: Web services description language 1.1. Retrieved August 15, 2004, from *http:// www.w3.org/TR/wsdl*

XML Core Working Group. (2000, October). W3C recommendation: XML version 1.0 second edition. Retrieved August 15, 2004, from *http://www.w3.org/TR/2000/REC-xml-20001006*

XML Protocol Working Group. (2003, June). W3C recommendation: SOAP version 1.2 parts 0-2. Retrieved August 15, 2004, from *http://www.w3.org/TR/SOAP/*

Chapter XIII

Service-Oriented Agents and Meta-Model Driven Implementation

Yinsheng Li
Fudan University, China

Hamada Ghenniwa
University of West Ontario, Canada

Weiming Shen
Fudan University, China

Abstract

Current efforts have not enforced Web services as loosely coupled and autonomous entities. Web services and software agents have gained different focuses and accomplishments due to their development and application backgrounds. This chapter proposes service-oriented agents (SOAs) to unify Web services and software agents. Web services features can be well realized through introducing software agents' sophisticated software modeling and interaction behaviors. We present a natural framework to integrate their related technologies into a cohesive body. Several critical challenges with SOAs have been addressed. The concepts, system and component structures, a meta-model driven semantic description, agent-oriented knowledge representation, and an implementation framework are proposed and investigated.

They contribute to the identified setbacks with Web services technologies, such as dynamic composition, semantic description, and implementation framework. A prototype of the proposed SOAs implementation framework has been implemented. Several economic services are working on it.

Introduction

Web services are featured with application, platform, and provider independence. They provide an appropriate paradigm for implementing open large-scale application environments. These environments can be viewed as collaborative integration environments with services. Services are not treated as isolated and one-time affairs but rather as elements in an interactive and dynamic collaboration structure. Service collaborations within or across environments are modeled in terms of supported transactions and processes. These collaborations are subject to norms and protocols specified for business domains. Services are thereby orchestrated vertically within one or horizontally across multiple environments. As a result, an individual environment streamlines services in terms of provided transactions while preserving its function scope to be highly specific to the targeted user group. Multiple environments collaborate to extend their business chains. Web services have been supported by major IT players through their commercial platforms such as Microsoft's .NET (Trowbridge et al., 2003) and SUN's J2EE/SUN One (Sun Microsystems). They are also underlying technologies behind the currently promoted business initiatives such as HP's Adaptive Enterprise (HP) and IBM's On-Demand e-Business (Wainewright, 2002).

Software agents have been developed with sophisticated interaction patterns. They are efficient in enforcing automatic and dynamic collaborations. Agent orientation is an appropriate design paradigm for e-business systems with complex and distributed transactions, especially for Web services. In services realization, software agents are very instrumental to provide a focused and cohesive set of active service capabilities. We therefore envision a Web service-based environment as a collection of economically motivated service-oriented agents (SOAs). SOAs cooperatively or competitively interact to provide common services in one specified environment, such as brokering, pricing, and negotiation in an e-marketplace, as well as cross-enterprise environment, such as integration and cooperation in an electronic supply chain. The fundamental elements of such environments are services, where transactions are behavioral aspects of the services. Software agents dynamically implement services as functionalities and roles.

The primary objective of our work is to integrate software agents and Web services into a cohesive body that attempts to avoid the weaknesses of each individual technology, while capitalizing on their individual strengths. There are two observations that set the stage for SOAs' comprehensive technical solution. On one side, Web services paradigm is fast evolving and has been provided with plentiful industry-oriented technologies. These technologies support Web services to be deployed, published, discovered, invoked, and composed in a standard and consistent way. It has therefore attained advantages, in that it is business and application driven. On the other side, software

agent paradigm has attained advantages in software construction, legacy systems integration, knowledge-based reasoning, and transaction-oriented composition and semantics-based interaction. With the above observations, we have found agent modeling technologies could lead to an appropriate solution for Web services implementation framework. A coordinative technical strategy for SOAs is therefore proposed. Agent orientation technologies are applied for SOAs construction, knowledge representation, and interaction patterns. Web service-oriented execution, invocation, management, and communication technologies and protocols are complied for business-oriented functionalities and environments. This arrangement will enforce SOAs with essential advances both in market acceptance over software agents and interactive automation over Web services.

This chapter addresses critical challenges with SOAs as related to principles, technologies, implementation framework, and application domains. We propose and investigate incurred concepts, system and component structures, a meta-model, a meta-model driven semantic description method, an agent-oriented knowledge representation solution, and an implementation framework. They consequently contribute to current main setbacks with Web services technologies, such as dynamic composition, semantic description, and implementation framework. A prototype of the proposed SOAs implementation framework, SOAStudio (versioned 1.02 as of this writing), has been implemented. Several economic services are being developed, for example, (reverse) auction services for commodity markets, brokering services for healthcare environments, and security services for financial domains.

Web Services and Software Agents: State of the Art

Web Services Technologies and Frameworks

We believe that Web services could be explained technically as a computation paradigm for constructing systems by autonomous, loosely-coupled and collaborative software components with emphases on standard protocols in service-related activities like description, publishing, deployment, discovery, and invocation. W3C and OASIS are devoted to developing Web services-related technologies and standards. There are a number of companies; for example, IBM, SUN, and MSC have been involved in the technologies development, as well as frameworks and platforms releases.

Web services technologies have emphases on open protocols and business-centric commitments. These kinds of protocols are fast emerging. Examples for basic Web services operations include XML, UDDI (OASIS, 2002), SOAP (W3C, 2003) and WSDL (W3C, 2001). Examples for Web services orchestration include BPEL4WS (McIlraith & Mandell, 2002) and ebXML (OASIS, 2001). They have been enriched with business-centric considerations. For instance, ebXML has been instructive for Web services to enforce its process engineering, though, from workflow's viewpoint. It combines process-involved business and execution logics with two views (Business Operational View and Functional Service View). BPEL4WS has been submitted to OASIS for approval, and a few companies have committed to accept it. It distinguishes abstract processes

from executable processes. The former is for describing business protocols, and the latter is for executable processes to be compiled into invocable services. BPEL4WS has been complemented by process-enhancing specifications, for example, WS-Transaction (Cabrera et al., 2002), WS-Coordination (IBM, Microsoft & BEA, 2003), and those special for security and business policy, for example, WS-Trust, WS-SecureConversation, WS-SecurityPolicy, WS-Policy, WS-PolicyAttachment, and WS-PolicyAssertions (Microsoft Corporation, 2002).

There are two main application development frameworks supporting Web services, for example, J2EE and .NET. They both accept basic Web services protocols such as UDDI, SOAP and WSDL. The majority of application and development environments, for example, WebSphere and Tomcat, are accepting either of them. Web services have seen a few successes at the application level, for instance, through e-business. There have been a few examples of using .NET-based Web services (Microsoft, 2002). Sun Microsystems has helped FordFinancial design to build a platform infrastructure based on the Sun ONE framework. Amazon.com Inc. and Google have given developers and owners the ability to build applications and tools using their Web services APIs. However, there is still no common implementation framework for Web services to facilitate their functional elements and implementation processes.

Web Services Semantics and Software Agents

An essential aspect for Web services is semantic description in a way that makes them recognizable and interoperable. A basic service semantic model accommodates its profile and representation. The profile plays a central role in publishing and searching services to satisfy a case. It provides sufficient information for service requesters or brokers to categorize and decide whether it satisfies their requirements. The representation involves service execution processes and is intended for service usage, composition, execution, and management. The profile and representation work together to support and automate service-oriented operations. There has been an attempt (Andreas, 2002) to specify a software agent using standard markup languages. However, current Web services technologies have not incorporated sophisticated semantics in their profile and representation. For example, UDDI (OASIS, 2002) is a profile-based protocol for registration and search, and it only relies on predefined keywords, instead of semantics. E-Speak (Karp, 2000) has a base vocabulary to define basic attributes but with no semantics with them. WSDL (W3C, 2001) has accommodated service profile, process, and usage with no well-defined semantics. BPEL4WS (McIlraith & Mandell, 2002) was initiated for process representation and orchestration. It is capable of supporting runtime semantics based on its logical operations and representation, but its formal semantics are controversial.

Web services paradigm is also promoted for its nature to support dynamic, distributed, and reconfigurable business and application integration. Current efforts have been obtaining this goal through Web services choreography and orchestration. The applied approaches are mainly originated from traditional top-down workflow principles. They have not distinguished Web services as autonomous entities. These features have been overlooked since service-oriented interaction patterns are more suitable for bottom-up

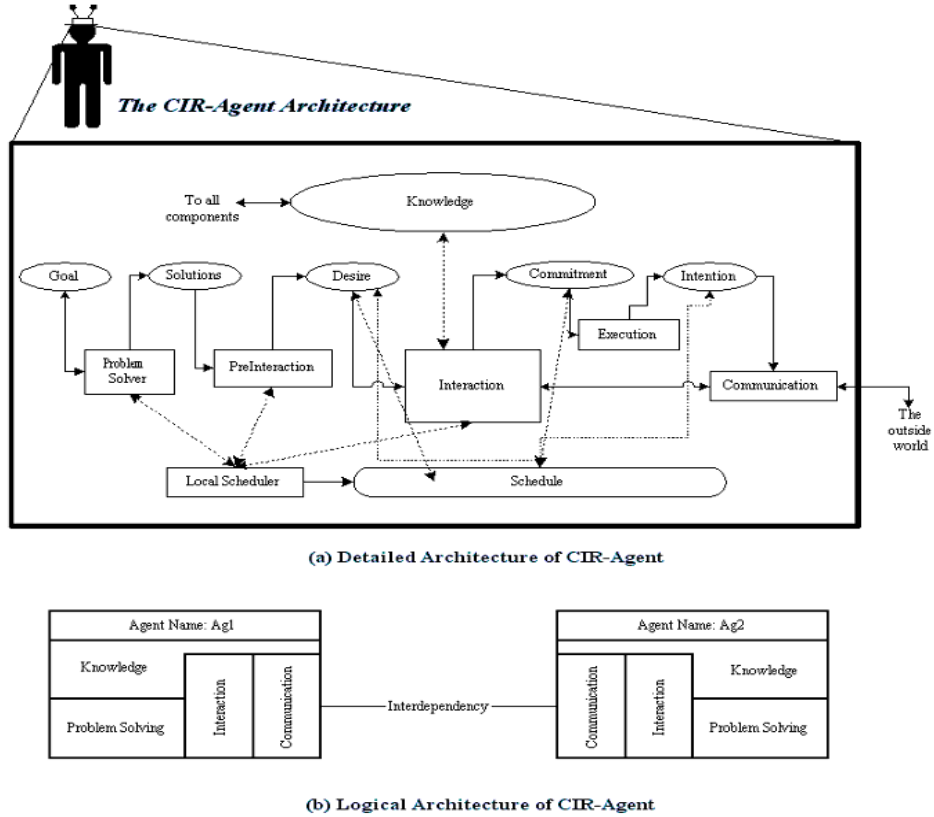
design and agent-based implementation, as we proposed in Li, Ghenniwa, and Shen (2003). Later, W3C (2003) introduced the concept of software agents as a foundation for Web services architecture specifications, called agencies of services. Further, W3C no longer separates Web services from provider agents. That implies that software agents are not used for services communication front ends, proxies, or as agencies. Rather, they are treated as unified entities that provide Web services. This approach goes along with our vision. We believe that agent-based services implementation, that is, service-oriented agents, can embody essential features and functions of Web services. The well-developed agent technologies for software construction, knowledge representation, and semantics-based interaction can thus be applied. It is a promising implementation framework for Web services.

Software Agents Models and Platforms

Here, we view "an agent" as a metaphorical conceptualization tool at a high level of abstraction (knowledge level) that captures, supports, and implements features that are useful for distributed computation in open environments. In our view, an agent is an individual collection of primitive components that provide a focused and cohesive set of capabilities. Figure 1 depicts the Coordinated Rational Agent (CIR-Agent) (Ghenniwa & Kamel, 2000). Each component of a CIR-Agent is associated with a particular functionality to support a specific agent's mental state as related to its goals. The basic components include problem-solving, interaction, and communication components. A particular arrangement (or interconnection) of the agent's components is required to constitute an agent. This arrangement reflects the pattern of the agent's mental state as related to its reasoning about achieving a goal. In a distributed context, for example, an e-marketplace, such agents play different roles (or provide services) and are able to coordinate, cooperate, and possibly compete with other agents including human beings.

Massive research efforts in software agents have yielded well-developed agent modeling and interaction technologies. The Foundation for Intelligent Physical Agents (FIPA) has been focusing on developing software specification standards for agent platforms and communication protocols. For example, the published communication language, FIPA ACL (FIPA, 2002), has been well known. An agent-based unified modeling language, AUML, is being developed by the FIPA Modeling Technical Committee (2003). There have been a few agent development frameworks and environments like JADE, FIPA-OS, and AgentBuilder (IntelliOne Technologies, 2001). JADE (TILAB, 2003) is a software framework to develop agent applications in compliance with the FIPA specifications (FIPA, 2002) for multiagent systems. It deals with all aspects external to agents that are independent of their applications, such as message transport, encoding and parsing, and agent life cycle. FIPA-OS (Emorphia Limited, 2003) is an experimental agent framework and component-oriented toolkit for constructing FIPA-compliant agents using mandatory components, components with switchable implementations, and optional components. In addition to supporting the FIPA interoperability concepts, it also provides a component-based architecture to enable the development of domain-specific

Figure 1. Logic and detailed architecture of CIR-Agent



agents, which can utilize the services of the FIPA Platform agents (that is, AMS, DF, and Dummy). AgentBuilder is a commercial integrated tool suite for constructing intelligent software agents. Agents constructed using AgentBuilder communicate using the Knowledge Query and Manipulation Language (KQML) (Finin, Labrou & Mayfield, 1997). AgentBuilder consists of two major components: the toolkit and the runtime system. The toolkit includes tools for managing the agent-based software development process, analyzing the domain of agent operations, designing and developing networks of communicating agents, defining behaviors of individual agents, and debugging and testing agent software. The Run-Time System includes an agent engine that provides an environment for execution of agent software.

There are attempts to develop ontology representation languages, for example, DAML+OIL (Richard & Deborah, 2001), Ontology Web Language (OWL) (W3C, 2003), and KIF (Genesereth & Fikes, 1992). A couple of ontology-level languages have been applicable for Web services, for example, DAML-S (OWL-S) (The DAML Services Coalition, 2002). The ontology-based comprehensive descriptions are supporting service-oriented registration, discovery, interaction, and collaboration.

However, market facts have not favored agent-oriented applications in that these technologies are mostly associated with academic researches. It is still far from successful commercialization despite that there have been a few attempts to put software agents into e-business environments (Bjornsson et al., 2002; Chavez & Maes, 1996; Ghenniwa, 2001; The Intelligent Software Agents Lab, 2001). Web services provide an application carrier for software agent technologies to be more market-driven and business-centric.

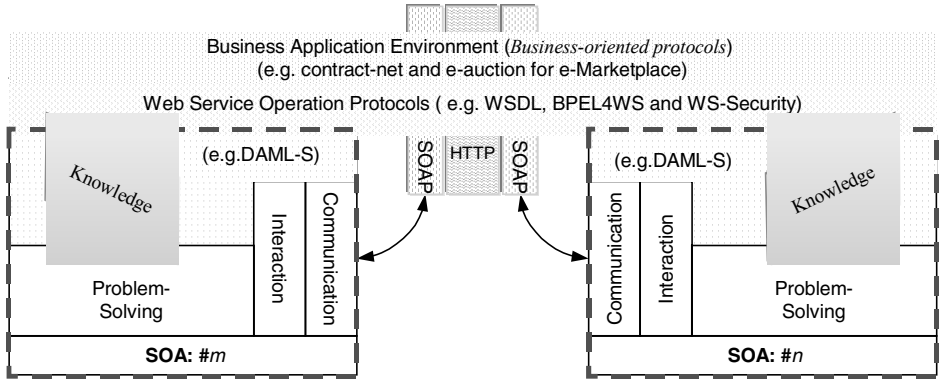
Service-Oriented Agents Framework

The concept of service-oriented agent (SOA) is to capture and model the intended functionality of the service into autonomous and dynamic interactions. A similar approach has recently been adopted by W3C (2003), in which an agent is defined as "a program acting on behalf of another person, entity, or process" while "specifically eschews any attempt to govern the implementation of agents." We strongly believe that SOAs could be one of the essential evolutions of Web services at functional entities, instead of simple interaction delegations or communication proxies. In system design, SOAs comply that Web services are designed by service-oriented methodology. Namely, the components are deemed service-oriented agents. The objective is for agents to account for the interaction aspect while enhancing services at their behavior.

Technically, the applied principle for SOAs is to exploit agent-oriented software construction, knowledge representation, and interaction methodologies and adapt them to service-oriented business transactions and application environments through specified Web services protocols. The major efforts toward this end include determining and coordinating technologies from Web services and software agents. As mentioned, Web services have proved to be prosperous in e-business applications while still undergoing fundamental setbacks in semantic integration, interaction behaviors, and implementation framework. Software agents have not scored in commercial applications while they could naturally contribute to the above setbacks. We have set up a coordinative technical framework for SOAs based on our observations.

The proposed technical framework accommodates multiple technical stacks from agent-oriented construction and service-oriented operation to business-oriented application environments. Figure 2 illustrates the scheme in terms of six layers, that is, business application environment, Web service operation protocols, knowledge representation, function entity, interaction, and transportation layers. At the layer of business application environment, for example, an e-marketplace or supply chain, service-oriented and business-centric protocols for transactions are applied for SOAs to collaborate with each other. They could be contract net, e-auction, and so forth. At the layer of services operation, service-oriented operation protocols, such as UDDI, WSDL, and BPEL4WS,

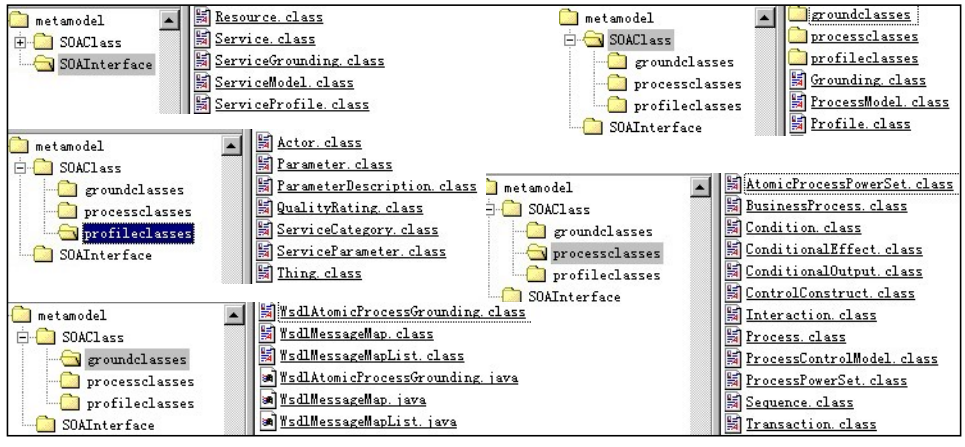
Figure 2. Technical framework for service-oriented agents



are applied to incorporate incurred considerations by the objective business transactions. At the layer of knowledge representation, the ontology-level language DAML-S and agent-oriented description and knowledge modeling technologies are applied to support SOAs autonomous behaviors. At the layer of function entity, the agent-oriented modeling and software construction technologies, for example, CIR-Agent and AUML, are applied to build the entity. The presented SOA modules enforce service abilities and interaction patterns in that the interaction components handle the conversations among SOAs, the problem-solving component provides the core computation function, and the communication component works with service-oriented messaging mechanism. At the layer of interaction or messaging, service-oriented SOAP or otherwise specified XML-based protocols are applied to implement SOA-oriented interaction performatives. Finally, at the layer of transportation, the basic Internet communication protocols such as HTTP, FTP, or HTTPS are applied to get through the data.

Knowledge representation is the key to coordinate technologies from two areas. On one side, knowledge base is a global component in an SOA software entity. The agent-oriented knowledge representation technologies qualify the SOA to support the communication, interaction, and problem solving components to analyze, determine, compute, and communicate. On the other side, the knowledge is established with ontology-level description languages and composed of semantic descriptions of itself, its tasks, and surrounding world. These descriptions could also represent what is involved in service-oriented protocols. This proves to be feasible since the current ontology supporting languages, for example, OWL and DAML-S (The DAML Services Coalition, 2002), have basic considerations to provide sufficient information to support agent-oriented interaction semantics and service-oriented discovery, composition, and execution.

Figure 3. An SOA meta-model for current prototype

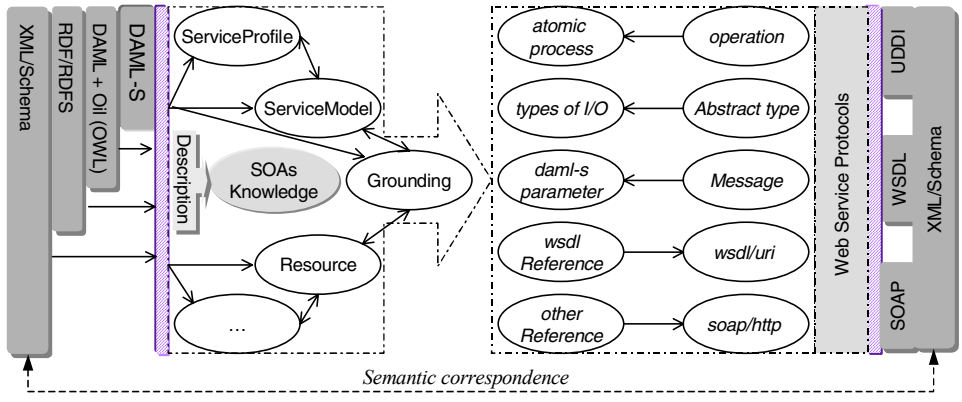


Meta-Model Driven SOA Implementation Framework

The concept of SOA is viewed as a computational model that enables a designer to capture and represent complex applications in open environments, for example, e-marketplaces, as a software artifact independent of the target framework. A meta-model is essential for SOAs to facilitate systems implementation and integration. This meta-model is mainly derived from concepts of ad hoc semantic Web language, that is, DAML-S (OWL-S). It expresses and relates fundamental SOA classes and operations concepts and is treated at two levels: (i) UML-based model for service capabilities and processes and (ii) agent-oriented model for service interactions (cooperative or competitive). As illustrated in Figure 3, a preliminary model for SOAs is currently composed of two main packages, SOAInterface and SOAClass. Five classes have been identified under the SOAInterface, that is, Resource, Service, ServiceGrounding, ServiceModel and ServiceProfile classes. Three classes, that is, Grounding, ProcessModel, and Profile, and three subpackages; that is, profileclasses, processclasses, and groundclasses are under SOAClass. There are further classes respectively under these subpackages.

SOAs use ontology-level languages to describe designated services, build their own knowledge, and support service-oriented composition and execution. The procedure is started with the semantic descriptions of the related services in terms of the proposed meta-model. The acquired SOA descriptions are further arranged to build SOA knowledge. An SOA's knowledge is equipped with a rule base, which incorporates execution logics and business logics into decision-supported and task-oriented rules. A consistent analyzer runs over the rule base to support the other SOA modules. SOAs, therefore, apply reasoning principles and accomplish automatic interaction patterns. Business-centric commitments and service-involved registration, discovery, execution, orchestra-

Figure 4. Meta-model-based SOAs description, knowledge, and their interlink with Web service protocols

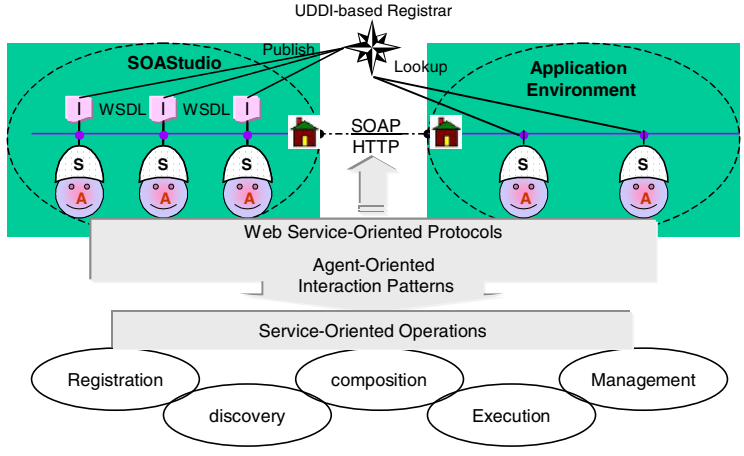


tion, and monitoring are achieved by this way. The agent-oriented information sensing and handling mechanisms enforce timely update of the descriptions and knowledge of SOAs.

There is a concern that the applied representation technologies should account for efficient exchanges between SOA knowledge and service-oriented protocols. The ontology-level language that SOAs use has addressed this concern (Li et al., 2003). It coordinates knowledge representation with Web services protocols through essential semantic correspondences between them. The technical scheme is illustrated in Figure 4. SOA description is built on the ontology-compliant meta-model. The interrelated constructs of the meta-model express the SOA and accommodate its abilities and usage. The description is supported by a variety of languages with accrued semantic schemas and expressiveness, for example, XML/Schema, RDF/RDFS, DAML+OIL/OWL, and DAML-S. SOA knowledge is derived from and aware of the description. From the other side of Web services environment, there are a variety of protocols, for example, UDDI, SOAP, and WSDL, enforcing Web services operations. Such operations include registration, discovery, execution, and composition. The interlink between the description and protocols is feasible since the meta-model of the former can also account for those elements of the latter. It is translatable since XML/Schema is the foundation for both of them. SOA knowledge is thereby feasible to efficiently handle service-oriented operations.

As mentioned earlier, an efficient and standard implementation framework is still undergoing with Web services. SOAs, however, have addressed this challenge through their associated frameworks and methodologies, for example, an implementation frame-

Figure 5. SOAs-based implementation framework

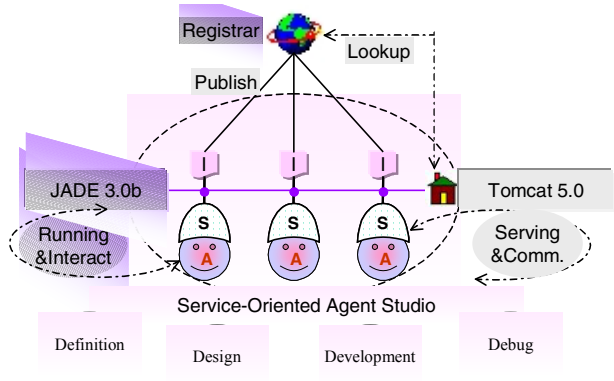


work in that a semantic meta-model, a CIR-Agent compliant software construction methodology, and a coordination technical framework have been present. As described in Figure 5, SOAs are designed to function as Web services while being built as software agents. They achieve service-oriented operations by agent-oriented interaction patterns. The ontological descriptions of them are organized into SOAs' knowledge modules (caps with the letter S). SOAs can therefore recognize and deal with Web service-oriented protocols, as discussed earlier. For example, they generate and expose WSDL interfaces (boxes with the letter I), publish and look up through UDDI-based registrar, and interact through SOAP and HTTP. Being built as Web applications, SOAs generally serve up behind Web servers to take advantages of Web technologies.

Prototype Implementation: SOAStudio

A prototype called SOAStudio (SOAs Studio, versioned 1.02 as of this writing) has been developed in terms of the proposed implementation framework. The studio provides platform-related components to implement SOAs and their working systems (for example, e-marketplaces), including development workspaces and mechanisms as well as basic application programming interfaces and examples. As illustrated in Figure 6, service-oriented agents are the objects under development. They are based on the proposed meta-model, and have agent constructs in function entity and DAML-S description to build agent knowledge. The extracted WSDL interfaces are published through UDDI-compliant registrar. The registrar is also an SOA and offered as a platform facility. SOAStudio considers six functionalities for SOAs definition, design, development, debugging, execution, and management. The first four of them are functional with SOAStudio. The execution and management environments are implemented by integrating JADE and Tomcat. SOAs are executed in JADE while served up via Tomcat. They are

Figure 6. System architecture for SOAs studio



therefore applicable for both FIPA ACL-compliant communication, usually within JADE platform, and SOAP-compliant communication, usually remotely.

As shown in Figure 7, SOAStudio is designed with a unified graphical user interface for users to interactively access system functionalities. The SOAs runtime environment includes an execution and management platform. SOAStudio integrates JADE and Tomcat by interacting with and managing their consoles, APIs and outputs. The unified interface provides users with four functional workspaces, that is, SOA design workspace, SOA development workspace, SOA debugging workspace, and SOA runtime environment. The design workspace is achieved by SOAs definition and project management functions. The definition function collects information about ongoing projects and SOAs. This workspace covers complete items for DAML-S service properties in terms of profile, process, and grounding. The development workspace is achieved by functions of making descriptions and agent files based on the meta-model and user inputs. The description files include DAML-based semantic description files and associated WSDL-based files. Agent-oriented class files are generated based on JADE-applicable agent classes with reference to the CIR-Agent structure. Developers can further customize and improve the codes to attain their specified SOAs with specified characteristics. The debugging workspace is composed of a building tool, an operation and output window, and a view tool. The building tool is provided to generate executable byte-stream classes on the generated agent files. The operation and output window monitors the performance of the studio and integrated platforms through operation log, outputs, and error reports. The view tool is used to check generated packages, descriptions, codes, and project data. Developers can complement pertinent resources to make the generated agent functional and refined. The runtime environment is achieved by the integrated JADE platform and Tomcat application server. They together provide execution, monitoring, and management environment for SOAs.

The SOAStudio integrates the JADE platform (TILAB, 2003). JADE supports a distributed environment of agent containers, which provide a runtime environment optimized to allow several agents to execute concurrently. This feature has been utilized to create

several concurrent market sessions, such as commodity and auction sessions. A complete agent platform may be composed of several agent containers. Communication in JADE, whether internal to the platform or external between platforms, is performed transparently to agents. Internal communication is realized using Java Remote Method Invocation to facilitate communication across the e-marketplace and its market sessions. External non-Java based communication between an e-marketplace and its participating organizations are realized through the Internet InterOrb Interoperability Protocol mechanism or HTTP. JADE provides the support for standard FIPA ontologies and user-defined ontologies.

Although the implementation takes advantage of the JADE platform and its supporting agents, such as a nameserver and a directory facilitator, the architecture of the SOAs is based on the CIR-Agent model (Ghenniwa & Kamel, 2000). Java features, such as portability, dynamic loading, multithreading, and synchronization support, make it appropriate to implement the inherent complexity and concurrency in a distributed environment, for example, e-marketplace. These features are also instrumental for executing CIR-Agents in parallel. The design of each SOA is described in terms of its

Figure 7. Service-Oriented Agent Studio (SOAStudio)

knowledge and capabilities. The agent's knowledge includes the SOA's self-model, goals, and local history of the world, as well as a model of its acquaintances. The agent's knowledge also includes its desires, commitments, and intentions as related to its goals. Implementation of the communication component takes advantage of JADE messaging capabilities. It is equipped with an incoming message inbox, whereby message polling can be both blocking and nonblocking, and with an optional timeout mechanism. Messages between SOAs are based on the FIPA ACL. The agent's reasoning capabilities include problem solving and interaction devices. The problem solving of an SOA is implemented through the use of complex behaviors. Behaviors can be considered as logical execution threads that can be suspended and spawned. The SOA keeps a task list, containing active behaviors. The problem-solving component varies from one SOA to another. The SOA behaviors can be classified as follows: behaviors that are concerned with distributed services, such as a market-registry service, advertisement service, mediation and auction service and behaviors that are concerned with providing business-specific services, such as selling and purchasing.

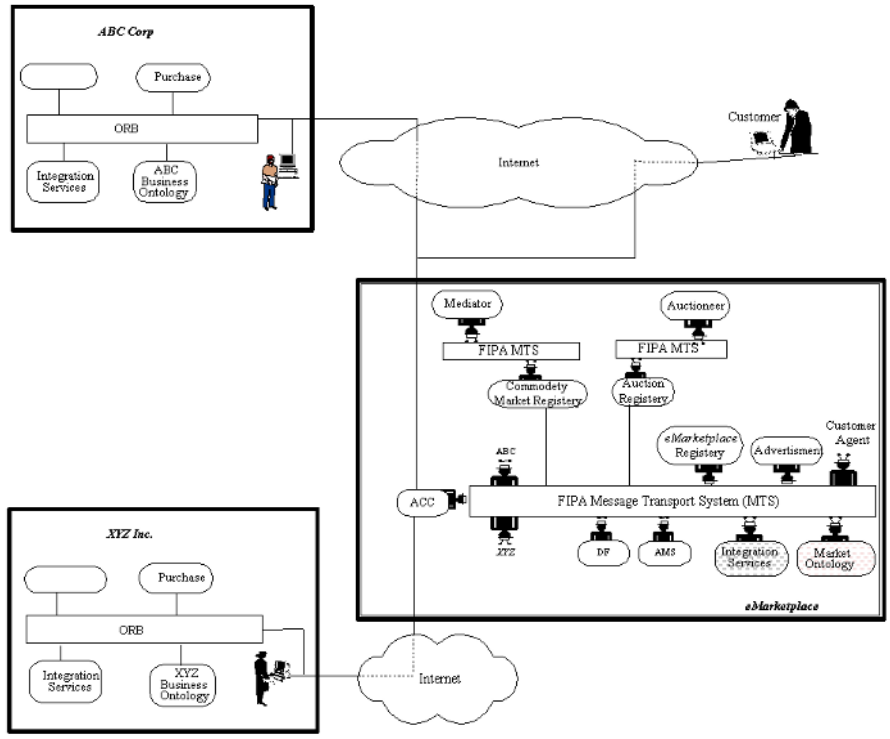
Case Study: e-Marketplace Services

The proposed SOA concept, related methodology and implementation framework as well as the SOAStudio prototype have been validated through a case study on e-marketplace services. The test environment is based on the business-centric knowledge-oriented architecture (BCKOA) (Ghenniwa, 2001). The BCKOA specifications provide the abstraction to support domain entities and applications independent of any specific technology. The main elements of BCKOA include domain services, integration services, and domain ontology. A key to BCKOA is a service-oriented CIR-Agent model.

As a distributed cooperative environment, an e-marketplace focuses on providing efficient business processes as well as value-added services such as community services, localization, customer relationship management, and authentication. It typically charges a subscription- or transaction-based fee as profits. A common functionality of an e-marketplace is to match buyers and providers of goods and/or services. The marketplace is therefore a broker among its members (buyers and providers). The valid buyers and providers could be businesses or individuals. The involved scenarios could be dynamic pricing for providers or optimizing procurement for buyers.

A BCKOA-applicable e-marketplace with SOAs is shown in Figure 8. ABC Corporation and XYZ Incorporated are virtual business entities registered with the e-marketplace for both purchasing and sales services. Both organizations use a BCKOA-based computation environment. Individual customers or business-entity personnel in the e-marketplace can participate in the market through their user interface agents. Similarly, each business-entity service is represented by an SOA in the e-marketplace. These SOAs provide intelligent and autonomous implementation for the business-entity services that might be based on legacy applications. For example, the ABC purchasing-service SOA represents the implementation of the business-specific purchases by ABC in the e-marketplace. Each user interface and business-entity SOA is registered in the e-marketplace. Thus, a user interface SOA can benefit from the market, business-specific,

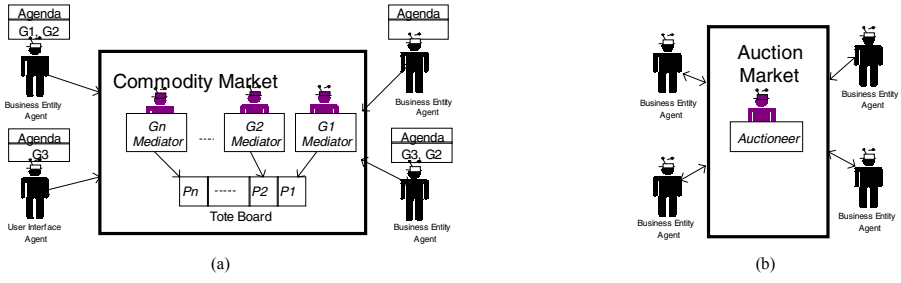
Figure 8. BCKOA-based e-marketplace with SOAs



and business-entity services by interacting with their representative SOAs. Each business-entity service must also be registered with a UDDI agent for the corresponding business-specific service. Each layer and its registry services are intended to provide some aspect of information about the e-business environment and enables an interested party to obtain information to potentially use offered services or to join the e-marketplace and either provide new services or interoperate as a trading partner with other business entities in that e-marketplace.

In the present case study, we experiment with commodity exchange and auction market structures, as shown in Figure 9, where customers and suppliers are brought together to trade with each other, and prices are set by the selected market structure. The trading behavior of the participant SOAs is also governed by the selected market structure. An individual customer is able to participate in the market through a dedicated user interface agent possibly assigned by the e-marketplace. Similarly, each participating business entity is assigned to a team of SOAs for the registered services and representative personnel who might have a direct contact with the market as well as with the customers. The market ontology provides a conceptualization of the domain at the knowledge level.

Figure 9. Logical design of (a) commodity market and (b) auction market session

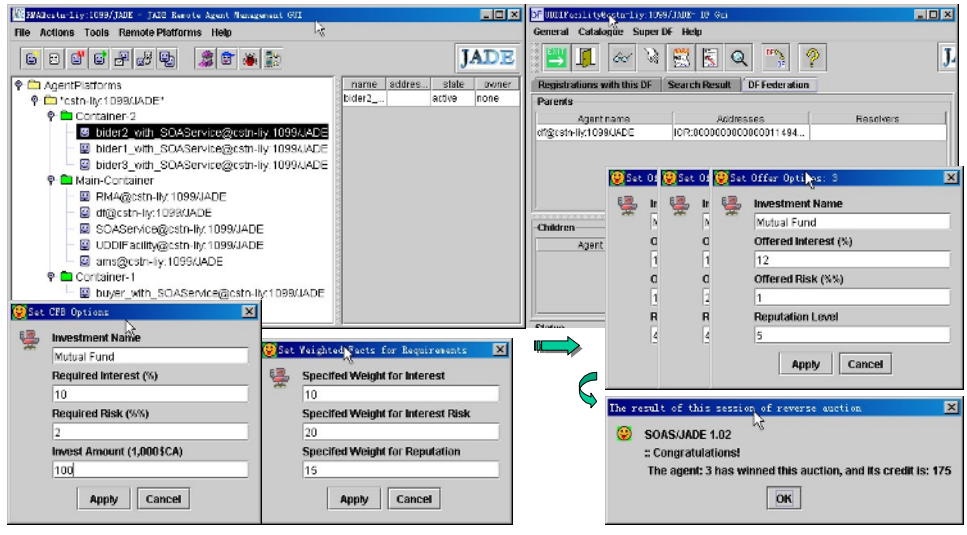


The current auction market structure can be applied either for seller-oriented auction service or for buyer-oriented reverse auction. Reverse auction allows multiple sellers to bid an item down to the lowest price for the buyer. A sample scenario of reverse auction could be found in Laskey and Parker (2000). The sense of competition heightens as suppliers compete in real-time by bidding lower as they see other offers. Buyers benefit from this service by assuring they receive the lowest price possible for various products and services they require. For the competition strategies, those for conventional auctions, for example, English, Dutch, Sealed-Bid, and Vickrey auctions, could also be processing protocols for reverse auctions. They are chosen on the basis of business-by-business analysis. We have adopted the Sealed-Bid strategy in the current prototype with higher bid (for example, lower overall charges) benefiting the buyer more.

One implementation consideration for the reverse auction is to assign the decision maker between the buyer and the auctioneer, that is, who determines the final winner. In our case, the auctioneer matches sellers and provides a recommended winner for the buyer to make a final decision for the auction session. A scenario of this e-marketplace service therefore comes up. A buyer submits a request for bids to a marketplace auctioneer through its user interface. The auctioneer starts with an auction session and sends requests for bids to all matching sellers. The sellers decide if they want to participate and submit bids through their interfaces. They may spell out some attached conditions related to their bids. For example, sellers may ask for a minimum price to be paid for the good. The auctioneer collects sells' bids, calculates their credits based on the buyer's preferences, and recommends a winner bid to the buyer. The best offer is determined on the buyer's concerns, since the definition of "goodness" may vary from one buyer to another. The concerns are therefore defined on the attributes of the requested good and their associated weights.

Figure 10 shows the performance of this case with SOAStudio. One buyer, three sellers (bidders), one auctioneer, and one UDDI registration SOAs are defined, developed, built, and deployed on the platform. The execution is done with JADE. The SOAs' interactions include registration with UDDIFacility, buyer's call-for-bid and credit calculation requirements settings, seller's offer settings, and winner recommendation to the buyer. During the auction, the auctioneer SOAService sends a call-for-bid to three bidders. The bidders set and submit offers. The auctioneer suggests the winner to the buyer. The case

Figure 10. Performance of a reverse auction with SOAStudio



does not get critical setbacks with the proposed principles except for programming and platform integration issues.

Conclusion and Future Work

Both Web services and software agents have been envisioned as appropriate computational paradigms for service-oriented distributed environments. However, they have gained different focuses and accomplishments due to their development and application backgrounds. Web services are motivated and associated intimately with businesses and applications. They have been promoted to incorporate inventive business and integration rules. They are therefore featured by business orientation and appropriate for dynamic e-business applications. Software agents have gained more research efforts along with artificial intelligence. They proved to be sophisticated in pertinent methodologies, for example, semantics-based reasoning and interaction-oriented software construction technologies. As discussed, Web services currently have challenges with semantic description and implementation frameworks. The current Web services efforts have not distinguished Web services right as loosely coupled and autonomous entities. These features have been overlooked since service-oriented interaction patterns are more suitable for bottom-up design and agent-based implementation. We have found

that Web service operations can be realized through software agents' sophisticated interaction behaviors. There is a natural way to combine them to integrate related technologies into a cohesive body that attempts to avoid the weaknesses of each individual technology, while capitalizing on their individual strengths.

This chapter proposes service-oriented agents (SOAs) to unify Web services and software agents and coordinate the related technologies. We have addressed critical challenges with SOAs in principles, technologies, implementation frameworks, and application cases. The incurred concepts, system and component structures, a meta-model, a meta-model driven semantic description, an agent-oriented knowledge representation, and an implementation framework are thus proposed and investigated. They consequently contribute to the identified setbacks with Web services technologies such as dynamic composition, semantic description, and implementation framework. A prototype of the proposed SOAs implementation framework called SOAStudio (versioned 1.02 as of this writing) has been implemented. The critical features of the SOAs paradigm, meta-model, semantic description, and implementation technologies have been demonstrated through the prototype. Several economic services are being implemented, for example, (reverse) auction services in commodity markets, brokering services in health care environments, and security services in financial domains. With convincing principles and experimental results, SOA is believed to be a promising methodology in service-oriented technologies and applications.

This chapter sets a stage for SOAs in paradigm and implementation technologies. Critical challenges can be anticipated ahead with the investigation, development, and applications. For example, a common collection of agent interaction patterns is a prerequisite to boost Web services-oriented dynamic businesses. We have obtained a preliminary UML-based meta-model. Their packages structure and class members need to be improved to incorporate agent-oriented paradigm. Semantic SOAs description technologies are also under development in parallel with their developments in Web technologies. The formal relationships between meta-model, description, agent knowledge, and Web service protocols have been identified as a key challenge with the proposed implementation framework. There are also imposed topics with technical stacks of SOAs since those related technologies are still evolving. The investigation on SOAs applications in a variety of distributed environments, such as e-marketplaces and supply chains, also makes a big difference since it could make direct benefits for industries.

References

Amazon.com Inc. Web services FAQ. Retrieved August 18, 2004, from *http://www.amazon.com/gp/aws/download_sdk.html/102-3008605-0441735*

Andreas, E. (2002). OntoAgent: A platform for the declarative specification of agents. In M. Schroeder & G. Wagner (Eds.), *Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web* (pp. 58-71).

Apache Software Foundation. The Tomcat 5 Servlet/JSP container. Retrieved August 18, 2004, from *http://jakarta.apache.org/tomcat/tomcat-5.0-doc/index.html*

Bjornsson, H., et al. (2002, January 2). FX-Agents description - White paper. Retrieved August 18, 2004, from *http://fxagents.stanford.edu/vision.php*

Cabrera, F., et al. (2002, August 9). Specification: Web services transaction (WS-Transaction). Retrieved August 18, 2004, from *http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/*

Chavez, A., & Maes, P. (1996). *Kasbah: An agent marketplace for buying and selling goods.* Proceedings of the 1st International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (pp. 75-90), London.

DAML Services Coalition. (2002, October 2). DAML-S: Semantic markup for Web services. Retrieved August 18, 2004, from *http://www.daml.org/services/daml-s/0.7/daml-s.html*

Emorphia Limited. (2003). Toolkit Overview. Retrieved August 18, 2004, from *http://www.emorphia.com/research/features.htm*

Finin, T., Labrou, Y., & Mayfield, J. (1997). KQML as an Agent Communication Language. In J. Bradshaw (Ed.), *Software agents* (pp.291-316). MA: AAAI/MIT Press.

FIPA. (2002, December 3). FIPA ACL message structure specification. Retrieved August 18, 2004, from *http://www.fipa.org/specs/fipa00061/SC00061G.html*

FIPA Modeling Technical Committee. (2003, July 15). Working documents. Retrieved August 18, 2004, from *http://www.auml.org*

Genesereth, M. R., & Fikes, R. E. (Eds.). (1992). *Knowledge interchange format, version 3.0 reference manual*. CA: Stanford University.

Ghenniwa, H. H. (2001). *eMarketplace: Cooperative distributed systems architecture.* Proceedings of the 4th International Conference on Electronic Commerce Research, Southern Methodist University, Texas.

Ghenniwa, H. H., & Kamel, M. (2000). Interaction devices for coordinating cooperative distributed systems. *Automation and Soft Computing, 6*(2), 173-184.

Google. Google Web APIs (beta). Retrieved August 18, 2004, from *http://www.google.com/apis/api_faq.html*

HP. HP adaptive enterprise glossary. Retrieved August 18, 2004, from *http://h71028.www7.hp.com/enterprise/downloads/res_glossary.pdf*

IBM, Microsoft, & BEA. (2003, September 16). *Web services coordination (WS-Coordination*). Retrieved August 18, 2004, from *http://www-106.ibm.com/developerworks/webservices/library/ws-coor*

Intelligent Software Agents Lab. (2001). RETSINA. Retrieved August 18, 2004, from *http://www-2.cs.cmu.edu/~softagents/retsina.html*

IntelliOne Technologies. (2001, December 17). Agent construction tools: Academic and research projects. Retrieved August 18, 2004, from *http://www.agentbuilder.com/AgentTools/index.html*

Karp, A. (2000, August 7). E-speak E-xplained. Retrieved August 18, 2004, from *http://www.hpl.hp.com/techreports/2000/HPL-2000-101.html*

Laskey, B., & Parker, J. (2000, July). Microsoft BizTalk Server 2000: Building a reverse auction with BizTalk orchestration. Retrieved August 18, 2004, from *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz/html/bizorchestr.asp*

Li, Y., Ghenniwa, H. H., & Shen, W. (2003, June). *Integrated description for Web service-oriented agents in eMarketplaces.* Proceedings of the Business Agents and the Semantic Web Workshop (pp. 11-17), Halifax, Nova Scotia, Canada.

McIlraith, S., & Mandell, D. (2002). *Comparison of DAML-S and BPEL4WS.* Stanford University: Knowledge Systems Lab.

Microsoft Corporation. (2002, April 10). Web services in action: Microsoft delivers software as a service with MapPoint .NET. Retrieved August 18, 2004, from *http://www.microsoft.com/presspass/features/2002/apr02/04-10mappoint.asp*

Microsoft Corporation. (2002, December 18). New group of specifications to build on industry work for Web services. Retrieved August 18, 2004, from *http://www.microsoft.com/presspass/press/2002/Dec02/12-18AdvancedSpecsBEPR.asp*

OASIS. (2002, July 19). UDDI version 2.04 API specification. Retrieved August 18, 2004, from *http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm*

OASIS. (2001, February 16). ebXML technical architecture specification v1.0.4. Retrieved August 18, 2004, from *http://www.ebxml.org/specs/ebTA.doc*

Richard, F., & Deborah, L. M. (2001, October). An axiomatic semantics for RDF, RDF schema, and DAML+OIL. Retrieved August 18, 2004, from *http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics.html*

Sun Microsystems. Ford Financial: J2EE technology-based architecture drives transformation across enterprise. Retrieved August 18, 2004, from *http://wwws.sun.com/software/sunone/success/Ford.pdf*

Sun Microsystems. Sun Open Net Environment (Sun ONE). Retrieved August 18, 2004, from *http://www.sun.com/software/sunone*

TILAB. (2003, March 19). Java Agent DEvelopment Framework. Retrieved August 18, 2004, from *http://sharon.cselt.it/projects/jade/*

Trowbridge, D., et al. (2003, June). Enterprise solution patterns using Microsoft .NET. Retrieved August 18, 2004, from *http://msdn.microsoft.com/practices/type/Patterns/Enterprise/*

W3C. (2003). Web Services Architecture. Retrieved August 18, 2004, from *http://www.w3.org/2002/ws/arch/*

W3C. (2003, June 24). SOAP version 1.2 part 1: Messaging framework. Retrieved August 18, 2004, from *http://www.w3.org/TR/soap12-part1*

W3C. (2003, August 18). OWL Web ontology language overview. Retrieved August 18, 2004, from *http://www.w3.org/TR/owl-features/*

W3C. (2001, March 15). Web Services Description Language (WSDL) 1.1. Retrieved August 18, 2004, from *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*

Wainewright, P. (2002, November 01). Loosely coupled Weblog. Retrieved August 18, 2004, from *http://www.looselycoupled.*

Section IV

Security in Service-Oriented Systems

Chapter XIV

Security in Service-Oriented Architecture: Issues, Standards and Implementations

Srinivas Padmanabhuni
Software Engineering and Technology Labs,
Infosys Technologies Limited, India

Hemant Adarkar
Ness Technologies, India

Abstract

This chapter covers the different facets of security as applicable to Service-Oriented Architecture (SOA) implementations. First, it examines the security requirements in SOA implementations, highlighting the differences as compared to the requirements of generic online systems. Later, it discusses the different solution mechanisms to address these requirements in SOA implementations. In the context of Web services, the predominant SOA implementation standards have a crucial role to play. This chapter critically examines the crucial Web services security standards in different stages of adoption and standardization. Later, this chapter examines the present-day common nonstandard security mechanisms of SOA implementations. Towards the end, it discusses the future trends in security for SOA implementations with special bearing on the role of standards. The authors believe that the pragmatic analysis of the multiple facets of security in SOA implementations provided here will serve as a guide for SOA security practitioners.

Introduction

Security is a fundamental issue of concern in computing systems. With the recent trends in distributed computing, primarily the emergence of World Wide Web (WWW) as a universal medium for conducting business, security has become critical in IT architectures. Successful Web security mechanisms like Secure Sockets Layer (SSL) have played a critical role in the emergence of WWW as a mainstream technology with wide acceptance.

In the context of distributed systems, SOA has caught the critical attention of both technology and business champions alike because of its promise in removing some of the hurdles in earlier models of distributed computing. Though SOA as a concept is not new, this promise is based on the open and loosely coupled nature of the newer SOA implementations.

In this chapter, we are concerned with multiple dimensions of security in loosely coupled SOA implementations. Web services, the most prevalent SOA implementation, represent an extension of the paradigm of Web. Web services represent applications that can be invoked over open networks using standard Web-based protocols. Other upcoming SOA implementations include Jini (Jini Spec, 2003), Open Grid Services Architecture (OGSA) (OGSA Spec, 2003), and so forth. We shall cover the various security requirements in SOA implementations, highlighting the differences from security requirements in generic online systems. We shall proceed to cover the different solution mechanisms to address these requirements in SOA implementations. Later, we shall explore the Web services security standards in detail. We shall then proceed to explore the common nonstandard security mechanisms of today, addressing Web services security. Towards the end, we shall present the future trends in security for SOA implementations including Web services.

Background

Since we are concerned with security of loosely coupled SOA implementations, we shall cover the generic security requirements and solutions in online systems, alongside the core concepts in SOA.

While online systems have been in use for the past few decades, the advent of the Web as a commercial medium has posed significant security challenges due to its public and open nature. Further, e-business and e-commerce has placed stringent security requirements due to the online transactions involved. Current security technologies in Web are able to handle and manage the expectations for e-commerce and e-business transactions. Security, in effect, broadly reflects a collection of security requirements to be satisfied. In this section, we point out the primary security requirements in online systems. Some of the typical security requirements of online systems are outlined below:

- *Confidentiality:* The confidentiality requirement states that any piece of information should not be understood by anyone other than the person for whom it was intended. Message privacy is a key requirement here.
- *Data Integrity:* The integrity requirement states that information should not be altered in storage or transit between a sender and the intended receiver without the alteration being detected.
- *Authentication*: The authentication requirement states that the sender and receiver should be able to confirm each other's identity and the origin/destination of the information.
- *Authorization*: The authorization requirement ensures that the sender has the required authority to perform the operation. This may range from permission to perform some action to permission for viewing some content.
- *Non-repudiation*: The nonrepudiation requirement ensures that the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information.
- *Privacy:* The privacy requirement is more general than the confidentiality requirement above. It also deals with the question of whether to trust the personal information with a Web site.
- *Trust:* This refers to the confidence in a person or a partner doing the transaction. This concept extends beyond trust in a person accessing an online service to even include participants in business-to-business transactions where trust may be used to refer to the adherence to the contractual agreements between the partners.
- *Auditing:* The ability to know who did what, when and where. This is a key requirement when it comes to detection of possible security breaches.
- *Availability*: The computing resources should be available for genuine users when they wish to access the resource. Denial of Service (DoS) attacks may cause lack of availability, and hence, there is a need to protect against such attacks.
- *Intrusion Detection:* The ability to detect when an unwanted user of an online system has entered the system and done some damage to the system.

Security Solution Mechanisms in Online Systems

Diverse security solutions and mechanisms have been designed and implemented for tackling online security requirements. In this section, we cover the different relevant solutions. This list is not meant to be exhaustive, however it will serve as the base for the discussion in the context of security solutions and standards for SOA implementations. The primary mechanisms of security employed in online systems are enumerated below:

- *Passwords:* A password refers to a unique secret series of characters which allows a user to access a computing resource. Ideally a password should be difficult to guess to prevent access to unauthorized users. It is the most common authentication mechanism in online systems.
- *Encryption:* Encryption is the most common security technique to ensure confidentiality in online systems. Essentially it refers to the process of taking a piece of data (called ***cleartext***) and a short seed string (a *key*) and producing an altered piece of data referred to as ***ciphertext*,** which is not understood by anybody who does not know the key. Decryption is the reverse process of converting the ciphertext to cleartext. Typically, encryption process relies on hard to emulate mathematical algorithms involving the key and the cleartext.

  Encryption algorithms can be divided into symmetric and asymmetric encryption algorithms. In symmetric algorithms, both the encryption and decryption keys are the same. Hence, they function on the basis of shared secret. In asymmetric algorithms, the encryption and decryption keys form a key pair, in which one key is a private key (which shall be kept a secret) and the other is a public key. If a piece of data is encrypted with the public key, it needs the private key to decrypt. Asymmetric methods distribution of the keys is easy, and hence, public key infrastructure relies on asymmetric methods. Encryption of messages ensures confidentiality by making it difficult to deduce the content of the original message from the encrypted message.
- *Access Control Lists:* These are generic formats of security information concerning permissions to access certain resources or to perform certain tasks. Most often, authorization is provided by usage of access control lists (ACL).
- *Hashing:* Hashing is another important technique used to ensure data integrity in online systems. The idea is to take an arbitrary-sized input data (referred to as a ***message***) and generate a fixed-size output, called a ***digest*** (or hash), such that it is nearly impossible to compute or guess the message from the hash. The hash of a piece of data can be used to verify the integrity after an online transfer by comparison with the recomputed hash of the transferred data.
- *Digital Signature:* Digital Signature is an important technique to ensure data integrity and nonrepudiation. Typically, the hash of a message is encrypted with the private key of an entity and is termed as the signature of the data. To ensure that the message received by the receiver is actually sent by the person who signed the message, the signature after decryption with the public key of sender should match the hash.
- *Digital Certificate:* Usage of digital signature in sending a message requires that the receiver knows *a priori* the sender's public key. This is a big constraint, and hence, it becomes important to make the public key available of the sender as part of the message to achieve flexibility. However, that opens up the requirement of the trust of the public key sent by the sender, whether it is genuine or not. Hence, to overcome these problems, specialized entities termed as certification authorities are entrusted with the task of signing the public key of senders and generate a

special form that can be sent along with a message. This signed form of representation of a public key is termed as a digital signature. By leveraging a third-party certification authority (CA), the problem of public keys is reduced to the receivers having to know the public key of the CA. Popular ways of broadcasting this information of public keys of CA entities include integrating them into the popular browsers or other online systems. Digital certificates are stored in standard formats like the popular X.509 Certificate format. Digital certificates are used for authentication and data integrity in public networks.

- *SSL:* SSL (Secure Sockets Layer) is a Web-based protocol that enables a secure connection between the client and the server. It is based on a series of exchange of keys (and the server digital certificate) between the server and the client to generate a session key that is used to encrypt all the following messages in the session. Typically as part of the protocol, the server certificate is requested by the client allowing the client to ensure communication with the right Web server. Thus, SSL enables channel encryption between the client and the server.

  Client-side SSL uses digital certificates of clients enabling them to prove their identity to the Web server. This personal certification attribute, or the client identification, is not very common at the moment due to the cumbersome process involved in maintenance of huge numbers of client certificates.

- *PKI:* Public Key Infrastructure (PKI) refers to a collection of authorities and a system for exchange of digital certificates to entities. A PKI set up typically includes a CA for generating, revoking, or maintaining the digital certificates. It also includes a registration authority (RA) for physically verifying the identity of a certificate requester using physical means like checking against an identity card before directing the CA to issue a certificate. CA uses the concept of Certificate Revocation Lists (CRL) for revoking inactive certificates.

- *Firewalls:* Firewalls are specialized security tools designed to protect an enterprise typically against attacks from the external network. All network traffic between the internal and external network is channeled through it, and the firewall allows only desired traffic as configured. Traffic from internal network to external network can also be filtered in the firewall. The conventional firewalls are typically based on the concept of packet filtering, and they operate on the network layer of the stack.

- *Code Signing:* A popular concept for ensuring security of downloadable code on the network is code signing. Any piece of code including Applets, Jar files, ActiveX controls, and so forth are signed before download is allowed. Thus, digitally signed code after download guarantees that the code really comes from the publisher who signs it and ensures that the content has not been corrupted or altered, so it is safe to run.

- *Sandbox model:* An alternative to code signing, the sandbox model, also applies to downloaded code on the network. Unlike the requirement of signing of every piece of code, it places restrictions on the capabilities of the downloaded code to

limit the harm it can do on the client machine. Thus, the sandbox model ensures safety to the client machine by restricting the capabilities of the untrusted code; for example, it is not allowed to look at the file system on the client machine.

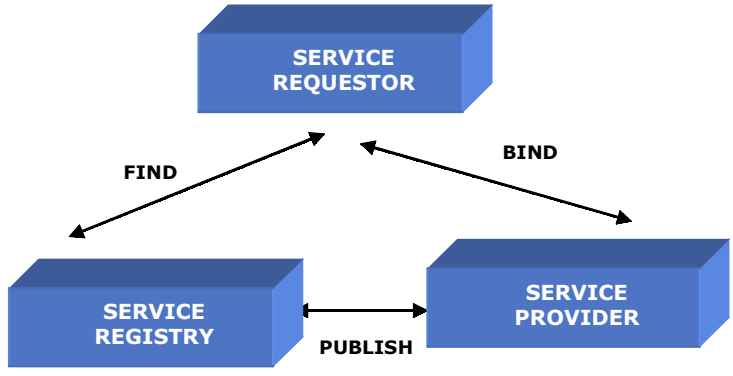
Service-Oriented Architecture

Research and development over the past few decades in distributed computing has resulted in the current day ideas in SOA implementations. SOA implementations revolve around the basic idea of a service. A service, unlike other previous concepts in distributed computing like objects or components, is more modular and self-contained. In raw terms, a service refers to a modular and self-contained piece of software, which has a well-defined functionality expressed in abstract terms independent of the underlying implementation that is accessible at a network point. Basically, any implementation of Service-Oriented Architecture has three fundamental roles: Service provider, Service requester, and Service registry and three fundamental operations: Publish, Find, and Bind (Figure 1). The service provider *publishes* details pertaining to service invocation with a service registry. The service requester *finds* the details of a service from the service registry. The service requester then invokes (*binds*) the service on the service provider. The role of service registry is sometimes also referred to as the service broker because it acts as a service broker between the requesters and providers.

SOA as a concept is not new to distributed computing. Earlier distributed architectural models based on CORBA, DCOM, Java RMI, and so forth had their basis in the SOA concept. However, these implementations were based on *tight coupling* between the service requesters and service providers. In these systems, there was a strict requirement of matching of data and protocol formats on both sides in order for the systems to interoperate. Further, tight coupling required significant changes in any dependent system when one system was changed, and hence, maintenance of systems in tightly coupled SOA implementations was costly.

On the other hand, a loosely coupled SOA implementation offers independence between the different participants so that each can act independently without requiring significant changes when one participant undergoes any change. This independence further extends the removal of strict matching requirements of data and protocol between the two systems. Such independence enables creation of flexible and adaptive distributed environments. Examples of such implementations of SOA include Web services, Jini, and OGSA. Unless otherwise mentioned, we shall use the term SOA implementations to refer to loosely coupled SOA implementations only. Web services represent standards-based functions accessible over a network with XML-based protocols. Jini (Jini Spec, 2003) is an architecture recommendation from Sun Microsystems for networks of devices interacting in a peer-to-peer loosely coupled fashion. OGSA (OGSA Spec, 2003) is an open loosely coupled SOA implementation of grid systems based on the concepts in Web services.

Figure 1. Service-Oriented Architecture



Security Requirements in SOA Implementations

SOA security requirements are more complex than that of online systems due to the following factors:

- Heightened security threats on account of easier access to implementations.
- Loosely coupled nature of interaction, requiring flexible ways of handling heterogeneous implementations.
- Dynamic interaction between services necessitates dynamic security mechanisms.
- Need to reuse existing security mechanisms in distributed systems because services are essentially wrappers over implementations.

Loose coupling security requirements in SOA will essentially translate to decentralization of conventional security mechanisms so that the interactions between service requesters and service providers carry the security information. These increased requirements necessitate additional security features including message level security, distributed credential management functions like single sign-on, message content inspection, interoperability of diverse security systems, and federation cum delegation requirements for trust and policies. Some SOA implementations like Web services advocate a recasting of the conventional security strategies and implementations for XML, and others reuse existing security concepts (for example, Jini uses Java RMI concepts).

Figure 2 lists some of the basic security requirements of an SOA implementation.

The primary security requirements in SOA implementations can be outlined as shown in Figure 2. A generic SOA implementation has been depicted in the figure, keeping in view different SOA implementations including Web services, Jini, and so forth. SOA implementations share the generic facets shown in the figure barring minor differences in details of execution of the processes between the three roles. While in some paradigms like Web services, the information retrieved about a service from the service registry could be plain interface information. In other paradigms like Jini, it could involve downloading of actual code proxies. We have highlighted the role of a service provider platform typically consisting of a development platform, combined with a deployment platform because the deployment platform is responsible for some of the enterprise grade requirements for service providers.

Online Security Requirements As Mapped to SOA

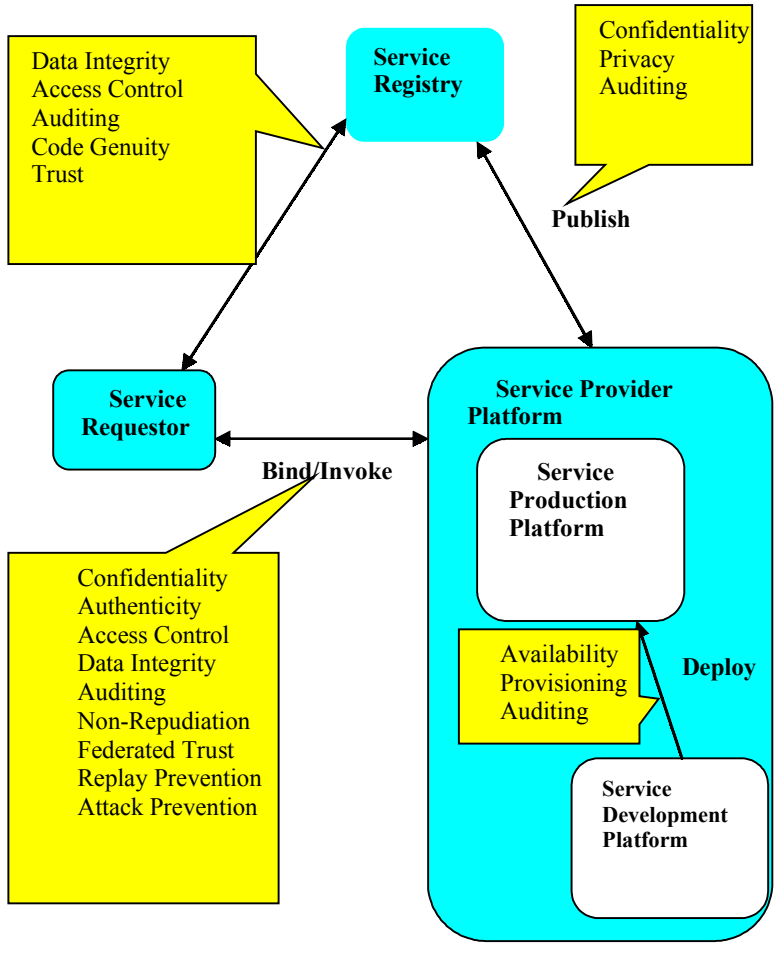
The primary security requirements in SOA implementations include the following:

- *Confidentiality:* The confidentiality requirement for Web services pertains to the requirement that a certain piece of information (XML Document) should not be understood by anyone other than the person for whom it was intended, while the rest of the document is left untouched. In conventional terms, this requirement applied to the whole of a piece of information. This necessitates partial encryption of the document.
- *Data Integrity*: The integrity requirement in Web services mandates that a portion of a piece of information (XML Document) should not be altered in storage or transit between sender and intended receiver without the alteration being detected, while other portions might be altered or in certain cases even be deliberately altered. For example, in a value added intermediary it is possible that extra information is added to the header of a message while the body is untouched. XML digital signatures can enforce such requirement of partial document integrity. In some other SOA implementations, it may be necessary to verify if a piece of code downloaded is correct. This is enforced by digitally signing the appropriate pieces of code.
- *Authentication:* In SOA implementations, it is necessary to have the capacity for one authentication system to interoperate with another authentication system for reasons of efficiency as well as usability. Further, more specifically with Web services, it might be necessary to accept credentials from outside an enterprise, for example, from a business partner. Such requirements can be met by standardized formats for authentication information accepted mutually or universally.
- *Authorization and Access Control:* The basic authorization mechanisms in online systems commonly prescribe static access control information where permissions are provided for static resources like file systems, systems, and so forth. In SOA implementations, there is a need to further specify operation level access privileges

(like permission to execute) dynamically because it may require invocation of operations by different kinds of clients at different times. Further, SOA implementations also need fine-grained specification of access control information related to restricting permissions to specific methods in a service, or a specific data item within a method, depending upon the requester.

- *Non-repudiation:* Besides capturing the fact that a certain piece of information was created or transmitted, nonrepudiation in SOA requires that any service invocation details be captured so that at a later time the requester cannot deny the invocation.
- *Privacy:* SOA implementations should have well-defined formalisms to use and disclose personal information provided by the service-requesting clients.

Figure 2. A bird's eye view of security requirements in service oriented architecture



- *Trust:* In SOA, a proxy on the service consumer is used to access services on the service provider. In some implementations, these proxies are downloaded dynamically. There is a need to implement mechanisms to trust the downloaded code and provide access to it to execute.
- *Auditing:* The ability to know who did what, when and where. This includes capturing of failed invocations of a service, faulty credentials, and so forth.
- *Availability:* The service should be available to a genuine service requester when requested. This is typically the responsibility of the service provider platform.
- *Provisioning*: Security provisioning refers to the process of administering the security information as part of the deployment. In SOA implementations, it is essential that devices or a hosting environment of a service be appropriately provisioned with the appropriate security credentials.

Additional Security Requirements for SOA

SOA poses additional security requirements that can be outlined below:

- *Single Sign-On:* Diverse service providers and service consumers have different authentication and authorization systems, and it is impractical for each system to maintain each other's authentication rights and access control lists. Single sign-on solutions remove the necessity of a universal credential by allowing credential mapping among many diverse systems. When one system authenticates a user, the state of authentication can be used by other systems without reauthenticating and with no change in the authentication mechanisms at other systems. Single sign-on can be implemented by standards for interoperating of security credentials.
- *Malicious Invocations:* The invocations of services may be done with malicious data, which may otherwise appear to be harmless to conventional solutions like firewalls, as input to the service. This malicious data may be in the form of spurious code being sent as part of a Web service request, or it can be a piece of untrusted code downloaded in a Jini environment. Appropriate code inspection technologies are required to identify the appropriate malicious code segments before it is acted on. Another approach to tackle the same is by secluding the running environment for the code execution, as followed by the sandbox model.
- *Repeated Invocations:* Owing to the ease of invocation of services with exposed interfaces, a repeated set of attacks on a service can occur leading to denial of service and loss of availability. This requires a specialized kind of firewalls with the capacity to diagnose the content of the request and examine method invocations in isolation. Code inspection of service requests can provide a clue about repeated requests and appropriate mechanisms can be used to avoid such attacks.
- *Delegation and Federation:* Delegation refers to the capability of a service or an organization to transfer security rights and policies to another trusted service or organization. Federation is a special case when the transferee service is a peer, not

a child. Delegation is a crucial component in SOA implementations like OGSA, where failure of a service should not let the whole federation collapse; instead, a delegation is done to another service. A related requirement is federated identity where users can single sign-on across heterogeneous hosts and Web services beyond one's enterprise. Standards and mechanisms to describe appropriate policies for federation and delegation are crucial to perform proper delegation and federation.

Security Standards and Solutions for SOA

Standards have special relevance to SOA security implementations, primarily Web services, as they form the crucial backbone for implementation of security solutions. Standards are crucial in any discussion of Web services security solutions. A plethora of security standards are being worked on at different standards bodies to enable faster adoption of Web services and grid technologies. Since Web services represent the predominant SOA implementation, we shall first cover the standards and implementations of Web services security before proceeding to discuss more generic SOA security solutions.

Web Services Security Standards

Web services security standards extend standard mechanisms like encryption, digital signature, and public-key infrastructure to handle XML and Web services specific nuances and problems. Some of the protocols are work in progress, and some have been standardized already. A top-level hierarchy of Web services standards is presented in Figure 3, and a detailed listing of the relevant key standards is shown in Table 1.
In the hierarchy shown in Figure 3, we have tried to highlight the fact that solutions to Web services security fall above the application layer in the conventional networking stack. Conventional network security protocols have been clubbed for the purpose of simplicity as the bottom-most layer. SSL, the predominant standard for Web security, provides point-to-point confidentiality and, hence, fails to address the situation where intermediaries are involved. Thus, SSL is not ideal for Web services.

Above this layer lies the layer of XML Signature (XML-Signature Spec, 2002) and XML Encryption (XML-Encryption Spec, 2002). Encryption and Digital Signature are key solution mechanisms for security in online systems. XML-Signature and XML Encryption are attempts to remap existing concepts in Web-based security to XML

message level security. In that sense, these protocols stress message-level security in contrast to session-level protocols in Web-based world. All the protocols at this level specify how to carry security data as part of XML documents. The ability to encrypt/sign only parts of XML documents is a critical part of security requirements for Web services as mentioned in the previous section.

Above this layer is XKMS (XKMS Spec, 2003), an attempt at extending PKI for XML-based Web services in order to promote widespread acceptability of XML-based security mechanisms. XKMS leverages the XML-Signature and XML-Encryption specifications.

Above this layer are the Web services security standards that leverage the lower level XML security specifications. An overlap is also observable between some key standards in this layer. The two standards of SAML and WS-Security are poised to become the popular *de facto* Web services security standards with wide acceptance. Other standards have not seen widespread acceptance. At the top lie the standards for federated identity of Web services. Federated identity is key to promote widespread acceptance of customer-focused dynamic Web services. In this section, we shall cover the key Web services security standards in greater detail.

XML Signature: In Web services, there is a need to partially encrypt the body or parts of a SOAP request to enable transmission with authenticity and integrity assured. Because of the involvement of such multi-hop data transfers in Web services, the original concept of digital signatures will not extend to XML-based content as it is based on the idea of getting signatures from the message digests of the entire document. Hence, intermediaries need mechanisms for development of complete trust of the handling of the content of messages keeping partial content intact. Such mechanisms have been provided in the XML Signature (XML-Signature Spec, 2002) specification.

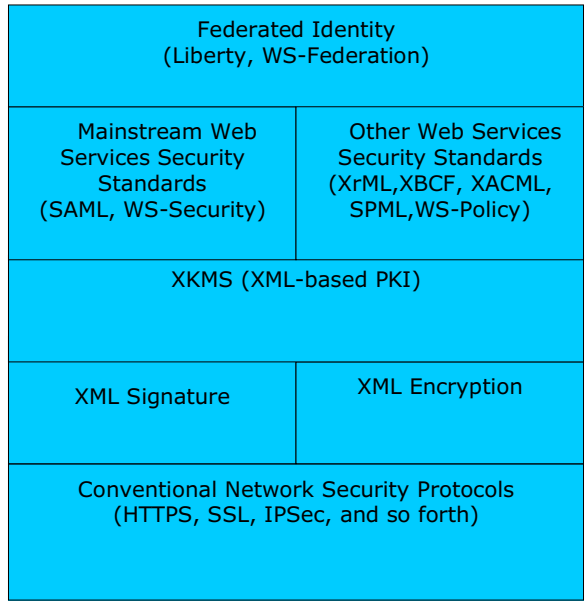
XML Signature defines an XML-compliant syntax for representing signatures over Web resources and portions of protocol messages and procedures for computing and verifying such signatures.

Such signatures will be able to provide data integrity, authentication, and/or nonrepudiation. In real-life scenarios, it is necessary that in the transmission route of the XML message, different parties sign different parts. XML Signature specification allows for this kind of signing. XML Signature validation requires that the data object that was signed be accessible. The XML Signature itself will generally indicate the location of the original signed object. This reference can:

- Be referenced by a URI within the XML Signature;
- Reside within the same resource as the XML Signature (the signature is asibling);
- Be embedded within the XML Signature (the signature is the parent-e*nveloping* form); and
- Have its XML Signature embedded within itself (the signature is the child-e*nveloped* form).

Typical computation of XML Signature of an XML document involves computing of the message digest of the given XML document. However, it is often necessary to understand there are many cases where seemingly dissimilar XML documents and nodes actually refer to the same document/node. A typical example is shown in Listing 1 (all listings are at the end of the chapter): All three of the above represent the same basic structure, that is, the structure 1. Thus, the idea in canonical XML is to obtain the core of any XML structure so that any two structurally equivalent XML documents are

Figure 3. Web services security standards stack



identical byte for byte in their core form. This core form is termed as the *canonical form* of an XML document. Canonicalization refers essentially to the process of conversion of any XML documents to its canonical form and is necessary for XML Signature computation. An example of enveloped XML Signature is shown in Listing 2. This example uses the DSAwithSHA1 algorithm to compute the signature, and the XML Signature also provides details of key values used to compute the signature.

XML Encryption: XML Encryption (XML-Encryption Spec, 2002) is a W3C standard for storing results of an encryption operation performed on XML data in XML form. XML Signature supports the concept of encrypting only specific portions of an XML document. This minimizes the encryption processing and leaves non-sensitive information in plain text form such that general (i.e., non-security-related) processing of the XML can proceed. This addresses the necessity of partial encryption of an XML document in case of involvement of intermediaries. A typical example of an XML-Encryption output is shown in Listing 3. It shows that only the credit card component is encrypted.

XKMS: This standard (XKMS Spec, 2003) is being developed to handle the infrastructure requirements for capturing the security information (values, certificates, and trust data) related to public Web services. It defines protocols for distributing and registering public keys suitable for use in conjunction with XML Signature. Essentially, it is attempting to extend concepts in PKI to XML-based interactions. It is being designed to support the key management functionalities including registrations, trust, and delegation when XML documents are involved. XKMS is critical for large-scale deployments of Web services for public consumers.

Table 1. A listing of key Web service security standards

Web Service Standard	Security Requirements Addressed	Proposed By	Status of Web Service Standard
XML Encryption	Message level Confidentiality	W3C	W3C approved
XML Signature	Message level Integrity, Nonrepudiation	W3C	W3C approved
XML Key Management System	XML-based PKI	W3C	W3C deliberations on
Security Assertions Markup Language	Single Sign-on, Authentication, and Authorization Interoperability	OASIS	OASIS approved
WS-Security	SOAP Message Security, Security credentials Interoperability	Microsoft, Verisign, and IBM	OASIS Spec. recently approved
XACML	Access Control and Policy Management	OASIS	OASIS approved
WS-Trust	Trust Management	Microsoft	No standardization
WS-Policy	Policy Management	Microsoft	No standardization
WS-SecureConversation	Secure Session Management	Microsoft	No standardization
XBCF	Biometrics	OASIS	OASIS approved
SPML	Service Provisioning	OASIS	OASIS approved
Project Liberty	Federated Identity	Sun and others	Project Liberty approved
WS-Federation	Federated Identity	Microsoft and IBM	None

SAML: Security Assertions Markup Language (SAML) (SAML Spec, 2003) is a standard ratified by OASIS as a high level XML-based standard for exchanging security credentials (assertions) among online business partners. The assertions could be authentication assertions (identity), attribute assertions (user limits and so forth), or authorization decision assertions (access control like read/write permission and so forth). It also specifies different protocols and profiles governing structure of SAML requests, responses, and mode of retrieval of assertions, for example, using SOAP over HTTP.

SAML does not prescribe a standard for authentication or authorization; instead, it offers a universal language to specify the information on authentication or authorization.

Hence, heterogeneous authentication mechanisms can interoperate using SAML. This way, SAML addresses the requirement related to single sign-on of Web services, too. SAML assertions can be signed using XML Signature. SAML is also the base for the federated identity system developed by Project Liberty. A sample SAML Response is shown in Listing 4. This example shows an authentication assertion and an attribute assertion which indicates the *membership* attribute as having the value *Gold*.

WS-Security: WS-Security (IBM-MS WS-Security Roadmap, 2002) is an overall framework for security for Web services. As part of the framework, the initial spec has been submitted to OASIS for ratification. Currently, WS-Security is in the final stages of public review and will be ratified soon. WS-Security has support from a critical mass of Web services vendors and will be a key standard for Web services security.

WS-Security provides a generic mechanism to attach a generic security token (public certificate, X.509 certificate and so forth) to SOAP messages in the header. The token takes care of many special variations including public certificates, shared tickets, and so forth. WS-Security defines how to attach signature and encryption headers to SOAP messages. Leveraging XML Signature in conjunction with security tokens provides message integrity, and leveraging XML Encryption in conjunction with security tokens provides message confidentiality. WS-Security allows interoperation of different existing security mechanisms like Kerberos, PKI, username-passwords, and so forth. A sample WS-Security SOAP message for passing username-password information is as given in Listing 5.

Federated Identity Standards: Federated identity is the ability to securely recognize and leverage user identities enabling single sign-on across disparate applications and hosts. Federation also allows an organization to securely share its confidential user identities with other trusted organizations with a single sign-on. Enterprises of today use multiple sources of identities including NT Domains, LDAP servers, RADIUS, and so forth, but none of them qualify for the federated identity system because of the heterogeneous nature of each of these protocols. There have been multiple attempts at creation of such a federated infrastructure by multiple vendors. The prominent among them are WS-Federation and Project Liberty. WS-Federation is a proposition from Microsoft/IBM combination based on the WS-Security specification. WS-Federation (WS-Federation Spec, 2003) describes how to manage the trust relationships in a heterogeneous federated environment including support for federated identities. Project Liberty (Project Liberty Spec, 2003) is a consortium of companies formed to provide a simplifying standard for federated identity, which will allow consumers to share credentials and enable single sign-on to disparate applications and Web sites. Its vision is to enable choice and convenience for consumers to access business services using any device connected to the Internet in a secure manner. Project Liberty has based its federated identity implementation on SAML as the core of the identity management system in combination with XACML, XKMS, and XML Signature. Project Liberty released its first functional specs in August 2002.

Current Web Services Security Implementations

In spite of the fact that all standards required for Web services security have not converged, most Web services security implementations have embraced the crucial Web services standards discussed in the previous section. We shall explore briefly the broad categories of Web services security solutions not necessarily based upon Web services standards.

XML Firewalls: In Web services environments, malicious attacks and DoS attacks present new challenges. XML firewalls (Quadrasis Firewall, 2003; Reactivity Firewall, 2003; Vordel Firewall, 2004; Westbridge Firewall, 2004) are a new generation technology, which operate above the conventional application layer unlike conventional firewalls that operate on the network layer. XML firewalls have the capability of examining an incoming SOAP request and taking an appropriate action based on the message content. Such content inspection is vital to prevention of malicious as well as DoS attacks. Further, XML firewalls can offer nonrepudiation mechanisms by providing audit trails of all service accesses.

XML Networks: Some Web services management vendors view that a network-based solution is better suitable for Web services, owing to the peer-to-peer nature of the paradigm. These solution vendors (Blue Titan Network, 2004; Flamenco Network, 2004) provide solutions that cater to various QoS parameters and sit at various network endpoints where the requests and responses from service consumers/providers respectively pass through in Web services invocations. Security is also a crucial function performed by these Web services networks.

Extension of EAI and Application Server Technologies: Existing enterprise application integration (EAI) products are enabling Web services today. They are able to provide security by extending their current security infrastructure, as it exists today. These vendors use some of the key existing Web-based technologies like SSL and X.509 certificates.

J2EE application server vendors base the Web services on current security infrastructures provided by J2EE platform itself. A leading number of J2EE application server vendors are working towards Web services standards like SAML, WS-Security, and so forth but have not come out with J2EE-based standard implementations of these standards due to the corresponding Java Specification Requests for Standardization (JSRs) that are still incomplete.

The .NET application server suite has been incrementally offering security support for Web services as outlined in IBM-MS WS-Security Roadmap (2002). Current versions fully support the WS-Security specification.

Security Infrastructure Vendors: Existing market leaders for Web-based security and PKI have made moves for extending their current implementations for Web services. These vendors have come up with suites of software products for securing Web services. Some of these products are offering support for SAML, WS-Security, and other standards. SAML is being adopted by a majority of identity management vendors (SAML Interop, 2002), while WS-Security is being adopted for Web services security. In the long

run, all vendors will offer entirely standards-based products once the dust on standards gets cleared.

Web Services Management Vendors: Many pure play and enterprise management vendors have jumped into the Web services management space. Some of these vendors are providing implementations based on the new Web services standards, while some are based on proprietary implementations. Most pure play or mainstream Web services management vendors have security as part of their management suite.

Security Solutions in Other SOA Implementations

Each SOA implementation has its own specialized security requirements, which may require specialized solutions.

Jini is essentially based on the idea of transfer of code from one entity to another. Hence, security of mobile code is important for Jini. Jini relies on the basic Java Security model and its extensions to fulfil the security requirements. The basic idea is to define a mechanism of trust for dynamically downloaded code and use it in combination with code-signing mechanisms. While code signing provides the integrity check, trust mechanism enables a decision point for execution of the dynamically downloaded code on the client.

OGSA has prescribed a standard for implementation of security in a service-oriented fashion. The recommended architecture for security in OGSA (Nataraj Nagartnam et al., 2002) is based around core Web services security standards. While it leverages SAML spec for the authorization service, it leverages WS-Security as the base for message-level security. The core recommendation of the security model for OGSA endorses the view of security requirements being provided by OGSA-compliant services. In that sense, certain core OGSA-compliant services form the infrastructure-level OGSA security services and are invoked for security needs. This model furthers the notion of a loosely coupled security model for the security requirements of a loosely coupled architecture model like OGSA.

Future Trends

SOA being a relatively recent trend in enterprise architecture, SOA implementations are still in infancy and have not matured yet. However, it is fast capturing the mindshare of enterprise architects with many enterprises announcing long term plans for migration to SOA. In due course of time, SOA implementations involving loose coupling will be pervasive across enterprises. In terms of trends in security for SOA implementations, we envisage the following trends in the future.

Standards Convergence and Maturity

The current standards stack in Web services security is a mix of standards under various stages of standardization: some (like XML Encryption) are fully ratified as standards; some (WS-Security) are nearing the end of the standardization process, while some (like WS-Federation) are proprietary and yet to be submitted to a standards body. Also, there is a clash in some competing security standards addressing the same requirements (for example, WS-Federation and Project Liberty both address federated identity).

Over time, it can be envisaged that the standards will converge to a core set of standards addressing Web services security. It is likely that two competing standards may be present for the same functionality with ways of interoperating. An example of such competing standards can be seen in SAML and WS-Security where there is some amount of overlap with each other, and WS-Security is working on a profile for including SAML assertions in WS-Security tokens. Some of the current security requirements like nonrepudiation, auditing, and so forth, which have not seen any standardization process yet, will be tackled by standards in the long run.

Standards Compliance for Security Products

Over time, the majority of Web services security offerings will be based on standards. This will lead to decreased cost of security administration. This is already evident from some category of products like identity management products, a majority of which have provided compliance with SAML (SAML Interop, 2002). In case of availability of multiple standards for the same implementation, products will offer interoperability support for multiple standards.

WS-I (Web Services Interoperability Organization) will have a key role in promoting standards adoption in products and vendors. WS-I is working on a security profile for Web services. This will be a key step in increasing standards compliance awareness among security vendors.

Federated Identity

Even though vendors and standards bodies have been quick in coming up with standards for federated identity in view of lack of convergence in standards and the enhanced skepticism, there is a lesser chance of federated identity-based business models being successful. Hence, federated identity as a concept will take more time to make inroads into enterprises.

XML-Based PKI

In view of the increased role of XML in SOA implementations in distributed systems, dealing with public keys will become a major scalability issue for SOA implementations.

The current PKI constituents like Registration Authorities and Certification Authorities will evolve to be compliant with XML-based PKI. Hence, in due course of time, XKMS will play a key role for the development of PKI.

New Generation Security Products

On account of the peer-to-peer nature of SOA implementations, specialized security intermediaries will be vital to managing the complexity of security requirements in enterprise grade and highly distributed environments. These specialized intermediaries will offload the security-related services away from the service consumers and service providers, thereby increasing scalability.

As part of this trend, some of the current day innovative SOA security products like XML networks and XML gateways will be commoditized owing to standardization of these innovative technologies as part of security best practices in enterprise architecture.

Component Security Models

Current component models like J2EE and .NET have varying levels of support for different Web services standards. The Java Community process (JCP) has currently admitted various specification requests to include various Web services security standards as part of J2EE spec. Until these are approved in the JCP, J2EE products will base their implementations of Web services security standards on proprietary mechanisms and APIs. In the .NET platform, the current support for WS-Security is proprietary, as the standard is only recently ratified. In the long run, it will be standards compliant.

Conclusion

In this chapter, we have examined the various security requirements as applicable to SOA implementations. As part of this exercise, we have seen how generic online security requirements map to SOA implementations. Further, the paradigm of SOA introduces new categories of security requirements.

Towards identifying potential solutions to the security requirements in SOA implementations, we have examined in detail the various standards and implementation mechanisms in Web services. Web Services being the predominant SOA implementation, these solution mechanisms guide us to appropriate security solutions for SOA implementations. We have shown the vital role of standards in Web services security implementations.

We also attempted a brief summary of different solution mechanisms in other SOA implementations. Since the SOA requirements for these implementations are varied, some of the concepts in Web services cannot be applied directly to these models. OGSA-based

grid solutions being based on Web services, solutions for OGSA security are directly dependent upon Web services security solutions.

Towards the end, we have tried to outline the future trends in SOA security on multiple dimensions, primary among them being the trends in standards. Over time, we feel that standards compliance will be commonplace among all the implementations.

We can see that standards have a key role to play in driving Web services security implementations. Over time, we feel that Web services security standards of WS-Security and SAML will occupy center stage for message-level security and identity management, respectively. These standards will provide flexibility as desired in Web services applications.

OGSA security architecture by its dependence upon core Web services architecture will be directly dependent upon the success of Web services security standards. Hence, the full realization of the security architecture for OGSA will take a somewhat longer time. A lot of work in security for Jini is still at a research stage, especially on requirements of trust management in networked computing and security of mobile code.

Further, we have seen that the loosely-coupled nature of SOA implementations necessitates a loosely-coupled approach to security, as is evident by the necessity of message-level security in Web services instead of the conventional channel-level security. We have also seen how SOA security implementation mechanisms advocate reuse of existing security infrastructure instead of fresh investment. All the standards for Web services security leverage existing security mechanisms and techniques in online systems and handle the extra requirements owing to usage of XML as the language.

Overall, security in SOA implementations is vital for success of SOA as a futuristic enterprise architecture paradigm.

References

Blue Titan Network. (2004, January 3). Blue Titan Network Director. Retrieved August 18, 2004, from *http://www.bluetitan.com/products/btitan_network.htm*

Flamenco Network. (2004, January). Flamenco Networks. Retrieved August 18, 2004, from *http://www.flamenconetworks.com/solutions/nsp.html*

IBM-MS WS-Security Roadmap. (2002, April 7). Security in a Web services world: A proposed architecture and roadmap. Retrieved August 18, 2004, from *http://msdn.microsoft.com/library/en-us/dnwssecur/html/securitywhitepaper.asp*

Jini Spec. (2003, September 12). Jini technology specification. Retrieved August 18, 2004, from *http://wwws.sun.com/software/jini/jini_technology.html*

Nataraj Nagaratnam, Jason Philippe, Dayka John, Nadalin Anthony, Siebenlist Frank, Welch Von, et al. (2002, July). OGSA security roadmap. Retrieved August 18, 2004, from *http://www.cs.virginia.edu/~humphrey/ogsa-sec-wg/OGSA-SecArch-v1-07192002.pdf*

OGSA Spec. (2003, April 5). Open grid services infrastructure. Retrieved August 18, 2004, from *http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf*

Project Liberty Spec. (2003, January 15). Project Liberty specification. Retrieved August 18, 2004, from *http://www.projectliberty.org/specs/archive/v1_1/liberty-architecture-overview-v1.1.pdf*

Quadrasis Firewall. (2004, January 2). Quadrasis EASI SOAP content inspector. Retrieved August 18, 2004, from *http://www.quadrasis.com/solutions/products/easi_product_packages/easi_soap.htm*

Reactivity Firewall. (2004, January 2). Reactivity XML firewall. Retrieved August 18, 2004, from *http://www.reactivity.com/products/solution.html*

SAML Spec. (2003, September 2). Security Assertions Markup Language (SAML). Retrieved August 18, 2004, from *http://www.oasis-open.org/committees/download.php/2949/sstc-saml-1.1-cs-03-pdf-xsd.zip*

SAML Interop. (2002, July 15). SAML interoperability event. Retrieved August 18, 2004, *from http://xml.coverpages.org/ni2002-07-15-a.html*

Vordel Firewall. (2004, January 2). Vordel XML security server. Retrieved August 18, 2004, from *http://www.vordel.com/products/xml_security_server.html*

Westbridge Firewall. (2004, January 2). Westbridge XML message server. Retrieved August 18, 2004 from *http://www.westbridgetech.com/products.html*

WS-Federation Spec. (2003, July 18). WS-Federation specification. Retrieved August 18, 2004, from *http://www-106.ibm.com/developerworks/webservices/library/ws-fed/*

WS-Policy Spec. (2002, December 18). WS-Policy Specification. Retrieved August 18, 2004, from *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wspolicyspecindex.asp*

WS-Security Spec. (2002, April 5). Web services security (WS-Security). Retrieved August 18, 2004, from *http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-security.asp*

XACML Spec. (2003, February 18). eXtensible Access Control Markup Language (XACML). Retrieved August 18, 2004, from *http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf*

XKMS Spec. (2001, March). XML Key Management Specification (XKMS). Retrieved August 18, 2004, from *http://www.w3.org/TR/xkms/*

XML-Encryption Spec. (2002, December 10). XML Encryption Syntax and Processing. Retrieved August 18, 2004, from *http://www.w3.org/TR/xmlenc-core/*

XML-Signature Spec. (2002, February 12). XML-Signature Syntax and Processing. Retrieved August 18, 2004, from *http://www.w3.org/TR/xmldsig-core/*

Source Code Listings

Listing 1. Cannonical Form Inputs

```
1. <node>b &amp;amp; c</node>
2. <node> b &amp;#x26; c</node>
3. <node><![CDATA[b&c]]</node>
```

Listing 2. Enveloped Digital Signature

```
<!-- Comment before -->
<apache:RootElement xmlns:apache="http://www.apache.org/ns/#app1">SOME
SIMPLE TEXT
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"></CanonicalizationMethod>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"></SignatureMethod>
<Reference URI="">
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"></Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315#WithComments"></Transform>
</Transforms>
<DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
<DigestValue>YNvvmanoIyMNl+33mqiZuJe9WlE=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>TrTeerc9ddqStQ0X/0XO/6G5k48kgUQtvRQofcbOZrJnYKyTJG9PX
Q==</SignatureValue>
<KeyInfo>
<X509Data>
<X509Certificate>
……..(Contents Shortened..)
</X509Certificate>
</X509Data>
<KeyValue>
<DSAKeyValue><P>
….(Contents Shortened).</P>
<Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</Q>
<G>9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxC
BgLRJFnEj6EwoFhO3
zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9
/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSo=</G>
<Y>
45T+wNtzv+XRinm6c/D/xb4DCcndZUtGeHva+0BbLBrIYHO2VN1mV1Sk1R4ThcPrjtx
Oa2Q4F6+O
MKIwSVIeCsk/2gUhHPNdBTEt+wEG7GpvO1QEE7i1k+AK8BhEzEAr7mUEh/7QhS6/
Kd+H0ZkLD/ZK
pTmYZnSP0EGVmscK0sY=</Y>
    </DSAKeyValue> </KeyValue> </KeyInfo> </Signature></apache:RootElement>
<!-- Comment after -->
```

Listing 3. Example of XML Encryption

```
<?xml version='1.0'?>
 <PaymentInfo xmlns='http://example.in/payments'>
   <Name>Srini<Name/>
   <CreditCard Limit='2,000' Currency='INR'>
     <Number>
       <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
        Type='http://www.w3.org/2001/04/xmlenc#Content'>
          <CipherData><CipherValue>A213C45D79</CipherValue>
          </CipherData>
       </EncryptedData>
     </Number>
     <Issuer>Online State Bank of the India</Issuer>
     <Expiration>03/04</Expiration>
   </CreditCard>
 </PaymentInfo>
```

Listing 4. Example of SAML Response with embedded Authentication and Attribute Assertions

```
<Response xmlns="urn:oasis:names:tc:SAML:1.0:protocol"
xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol" IssueInstant="2003-10-
16T13:04:03Z" MajorVersion="1" MinorVersion="0" Recipient="ravi"
ResponseID="7ea17dd3-655a-40e9-a890-ab548c0d71c1">
<Status>
<StatusCode Value="samlp:Success">
</StatusCode></Status>
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" AssertionID="dab9ae6d-
01a2-4122-a3a0-a2da221907e8" IssueInstant="2003-10-16T13:04:08Z"
Issuer="Srinivas" MajorVersion="1" MinorVersion="0">
<Conditions NotBefore="2003-10-16T13:04:03Z" NotOnOrAfter="2003-10-
16T13:06:03Z">
</Conditions>
<AuthenticationStatement AuthenticationInstant="2003-10-16T13:04:03Z"
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:unspecified">
<Subject>
<NameIdentifier>
Gold
</NameIdentifier>
<SubjectConfirmation>
<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod
>
</SubjectConfirmation>
</Subject>
</AuthenticationStatement>
<AttributeStatement xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Subject>
<NameIdentifier>Gold</NameIdentifier>
</Subject>
<Attribute AttributeName="Membership" AttributeNamespace="namespace">
<AttributeValue>Gold</AttributeValue>
</Attribute>
</AttributeStatement>
</Assertion>
</Response>
```

Listing 5. WS-Security example with username password credentials

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope
 <S:Header>
   <wsse:Security>
   <wsse:UsernameToken wsu:Id="MyID">
     <wsse:Username>Sam</wsse:Username>
     <wsse:Password>MyPassword</wsse:Password>
     <wsse:Nonce>FKJh...</wsse:Nonce>
     <wsu:Created>2001-10-23T09:00:00Z</wsu:Created>
   </wsse:UsernameToken> ..... </wsse:Security>
 </S:Header>
   <S:Body wsu:Id="MsgBody"> ..... </S:Body>
    </S:Envelope>
```

Chapter XV

A Service-Based Approach for RBAC and MAC Security

Charles E. Phillips, Jr.
United States Military Acadamy, West Point, USA

Steven A. Demjurian
University of Connecticut, USA

Thuong Doan
University of Connecticut, USA

Keith Bessette
University of Connecticut, USA

Abstract

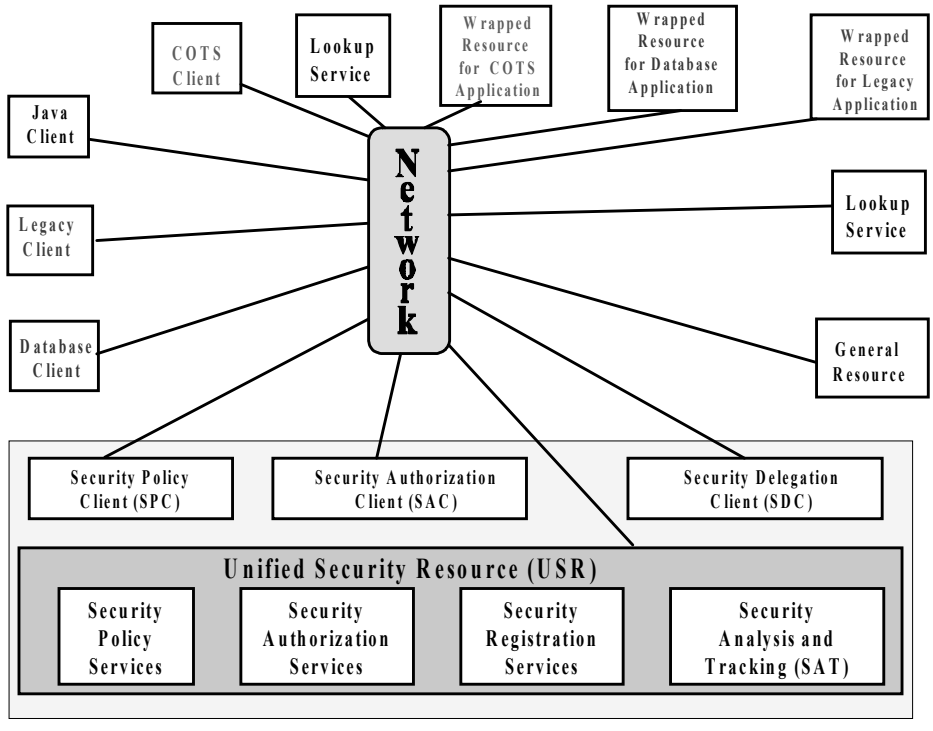
Middleware security encompasses a wide range of potential considerations, ranging from the ability to utilize the security capabilities of middleware solutions (for example, CORBA, .NET, J2EE, DCE, and so forth) directly out-of-the-box *in support of a distributed application to leveraging the middleware itself (paradigm) to realize complex and intricate security solutions (for example, discretionary access control, role-based access control, mandatory access control, and so forth). The objective in this chapter is to address the latter consideration: examining the attainment of advanced security capabilities using the middleware paradigm, namely, role-based*

access control (RBAC) and mandatory access control (MAC). The resulting security provides a robust collection of services that is versatile and flexible and easily integrates into a distributed application comprised of interacting legacy, COTS, GOTS, databases, servers, clients, and so forth.

Introduction

One challenge facing government and corporations today is to architect and prototype solutions that integrate new and existing software artifacts (that is, legacy applications, COTS, GOTS, databases, clients, servers, and so forth), facilitating their interoperation in a network-centric environment via middleware (collections of services), thereby providing the computing infrastructure to support day-to-day operations, as shown in the top portion of Figure 1. In these distributed collections of software artifacts, security

Figure 1. The security framework



must play a fundamental role, considered at early and all stages of the design and development life cycle. Middleware security encompasses a wide range of potential considerations, ranging from utilizing out-of-the-box security services of middleware platforms, that is, DCE (Open Software Foundation, 1994; Rosenberry, Kenney & Fischer, 1992), CORBA (Object Management Group, 2002; Vinoski, 1997; Yang & Duddy, 1996), DCOM/OLE (Microsoft Corporation, 1995), J2EE/EJB (Roman, 1999; Valesky, 1999), Jini (Arnold et al., 1999; Waldo, 1999), and .NET (Riordan, 2002; Sceppa 2002), to custom-built service-based solutions that realize complex and intricate security approaches (for example, discretionary access control, role-based access control, mandatory access control, and so forth).

In such a scenario, one can conceptualize each of the software artifacts in terms of *resources* that provide *services* (methods) for use within the environment, and as such, each artifact publishes an *application programmer interface (API)*. The problem with these APIs is that they contain all of the public methods needed by all users without regard to security. If one user (for example, a physician) needs access to a method (for example, prescribe_medicine) via a patient tool, then that method must be part of the API, and as such, the responsibility would be on the software engineer to ensure that the method is only accessible via the patient tool to users who are physicians and not all users of the patient tool (which may include nurses, administrators, billing, and so forth). Thus, in many applications, the ability to control the visibility of APIs (services) based on user role would be critical to ensure security.

Towards this end in this chapter, we present a service-based approach using middleware that unifies role-based access control (RBAC) and mandatory access control (MAC) into a security model and enforcement framework for a distributed environment comprised of interacting software artifacts (Liebrand et al., 2003; Phillips et al., 2002a; Phillips et al., 2002b; Phillips et al., 2003a; Phillips et al., 2003b). Our approach concentrates on the APIs of software resources, the services, providing the means for them to be customizable and restricted by time intervals, data values, and clearance levels to define the portions of APIs that can be invoked based on the responsibilities of a user role and the security level of the user. For enforcement, as shown in the bottom half of Figure 1, there is the Unified Security Resource (USR), which provides the security infrastructure via services: *Security Policy Services* to manage roles; *Security Authorization Services* to authorize roles to users; and *Security Registration Services* to identify clients and track security behavior. These services are utilized by a set of administrative and management tools, namely: the *Security Policy Client (SPC)* to manage user roles, the *Security Authorization Client (SAC)* to authorize roles to end users, and the *Security Delegation Client (SDC)* to handle the delegation of responsibilities from user to user. Our objective in this chapter is to explore in detail the middleware services that we have designed in support of RBAC/MAC security modeling and enforcement.

The remainder of this chapter has five sections. First, an overview provides background on the security modeling capabilities and features in our approach. Then, the middleware services and associated processing that supports RBAC/MAC security is examined. Next, the prototyping of USR and security tools (Uconn, 2003) is explored, which uses the technologies Jini 1.1 and Visibroker 4.5 to realize sophisticated and complex security capabilities. After that, RBAC/MAC security within other frameworks will be discussed.

Then, future trends are discussed with a focus on the support for security in CORBA, .NET, and J2EE. Finally, concluding remarks are presented.

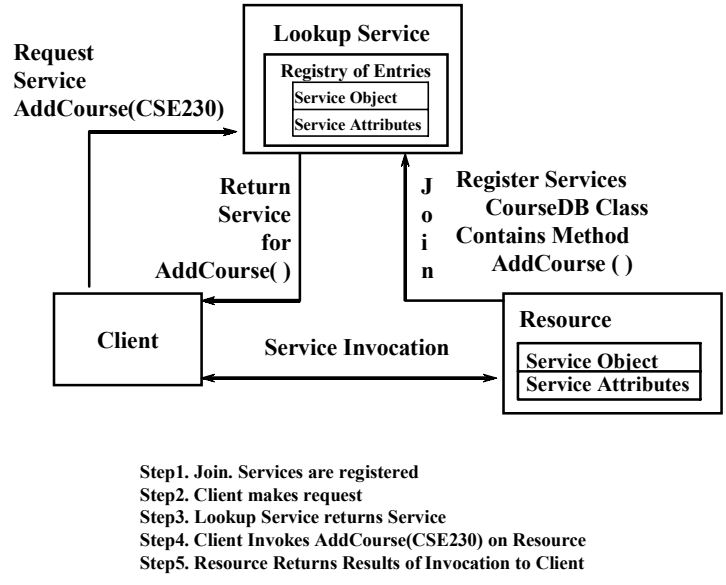
Background

This section presents background concepts of middleware that are important for understanding the chapter, which provide a context for the RBAC/MAC security model that serves as the basis of the work presented herein. A lookup service, as supported in middleware such as CORBA, Jini, DCOM, and so forth, is a clearinghouse for resources to register services and for clients to find services. A lookup service allows stakeholders to construct distributed applications by federating groups of users (clients) and the resources that they require (Arnold et al., 1999; Demurjian et al., 2001). With any lookup service, it is key that the services are registered, or they will not be readily available. Resources register services provided for use by a person, program (client), or another resource, including a computation, a persistent store, a communication channel, a software filter, a printer, and so on. Figure 2 illustrates the interactions of a lookup service, client, and resource in a distributed resource environment (DRE). In Figure 2, the CourseDB resource joins the lookup service by registering its services (methods). Each service consists of methods (for example, AddCourse) that are provided for use by clients (and other resources). Figure 2 also illustrates the steps that are taken when a client requests a service (AddCourse) from a resource (CourseDB).

The main limitation of the process as given in Figure 2 is that once registered, all of the resource's services are available to all clients; that is, there is no security. To integrate security into a service-based environment, we have developed a RBAC/MAC security model (Phillips et al., 2002a; Phillips et al., 2002b) with role delegation (Liebrand et al., 2003) that provides a means to formalize relevant middleware and security concepts and combine them into an approach that promotes secure policy definition and real-time access using security services via a Unified Security Resource (USR - see Figure 1 again). Since our emphasis in this chapter is on the security services, we provide only a brief summary of the security model as background.

To begin, to constrain access based on time, we define a *lifetime, LT,* as a time interval with start time (st) and end time (et) of the form (mo., day, yr., hr., min., sec.). Lifetimes will be utilized in the definition of the various constructs of the security model and by the USR security services to verify if access can occur at the current time. In support of MAC, we define s*ensitivity levels* unclassified (U), confidential (C), secret (S), and top secret (T) forming a hierarchy: U < C < S < T with c*learance (CLR)* given to users and *classification (CLS)* given to entities (roles, methods, and so forth). To characterize the distributed environment (top half of Figure 1), we define a distributed application as a set of unique *software resources* (for example, legacy, COTS, database, and so forth), each composed of a unique set of *services*, which each in turn are composed of a unique set of *methods*. *Methods* are defined by their name, LT of access, CLS level, and parameters; *services* and *resources* by their name, LT of access, and CLS level, where CLS

Figure 2. Join, lookup, and invocation of service



of a service is minimum (least secure) of its methods, and the CLS of a resource is the minimum of its services. Methods have LTs to indicate when they are available via a lookup service and have CLSs to denote the level of security classification that must be present in order for the method to be invoked.

To represent privileges, we define a *user role, UR*, to be a triple of name, LT of access, and CLS level, and maintain a list of all URs for an application. Next, a *user, U,* is uniquely identified by UserId (typically name), LT of access, and CLR level with an associated user list for the application. At security policy definition time, the services of USR are utilized by a security officer to define URs and once defined, to assign each UR a select set of methods that represent the privileges that a role is allowed. To further customize this process, we allow this assignment to be constrained based on a method's actual parameter values (called a *signature constraint, SC*) and limit when a method can be active for a UR in time (called a *time constraint, TC*). Thus, a *user-role authorization, URA,* associates a UR with a method M constrained by SC (what values can a method be invoked) and a TC (when can a method be invoked). Then, a *user authorization, UA,* associates a user with a UR and a TC (when can a user be authorized to a role). Finally, at runtime a *client, C,* is authorized to a user, identified by a unique *client token* comprised of user, UR, IP address, and client creation time. Given a client (an identifiable user) with a selected UR, in order for a lookup of a method (see Figure 2) to occur, there must be a number of dynamic security checks using USR to ensure that all of the defined security privileges are satisfied at runtime (enforcement) prior to the invocation.

To illustrate the concepts, consider a health care application could have URs for Nurse, Physician, Biller, and so forth, and these roles could be assigned methods for manipulating a resource Patient Record that has methods Record_Patient_History assigned to Nurse, Set_Vital_Signs assigned to Nurse and Physician, Prescribe_Medicine assigned to Physician, and Send_Bill assigned to Biller. Actual users would be assigned URs (for example, Steve to Physician, Lois to Nurse, Charles to Biller, and so forth), thereby acquiring the ability to invoke methods. There could also be limits placed on the methods assignment to a role to control values (for example, Nurse Lois can only access patients Smith and Jones for the Record_Patient_History and Set_Vital_Signs methods) or to control time (for example, Biller Charles can only access Send_Bill after patient has been discharged).

The USR Security Services

This section focuses on describing in detail, the Unified Security Resource (USR) as given in Figure 1, which consists of three sets of services: *Security Policy Services* managing roles and their privileges, *Security Authorization Services* to authorize roles to users, and *Security Registration Services* to identify clients and track security behavior. The USR is a repository for all static and dynamic security information on roles, clients, resources, authorizations, and so forth and is organized into a set of services, as given in Figures 3 and 4. These services are utilized by the various security tools (Security Policy, Security Authorization, and Security Delegation clients) (Figure 1) and by the resources that comprise the distributed application. In the remainder of this section, we begin by providing a detailed review of the different services of USR. We follow this introduction with an examination of the service interactions that demonstrate the security checks and verifications that are performed to ensure that a user playing a role is allowed to invoke a method at a particular time.

- *Security Policy Services* (Figure 3) are utilized to define, track, and modify user roles to allow resources to register their services and methods (and signatures) and to grant/revoke access by user roles to resources, services, and/or methods with optional time and signature constraints. These services are used by a security officer to define a policy and by the resources (for example, database, Java server, and so forth) to dynamically determine if a client has permission to execute a particular resource, service, or method under a time and/or signature constraint. There are five different services:
- The *Register Service* is provided to allow a resource to (un)register itself, its services, and their methods (and signatures), which is used by a resource for secure access to its services. The objective is to register the identifier, lifetime (LT), and classification (CLS) level for each resource, service, and method. Each resource can have its services and methods registered in two different ways. One approach has the resource controlling the registration process and registering itself. A second approach puts the registration under the control of the security officer using the

Security Policy Client (SPC) (see Prototyping section). Both of these approaches utilize the Register Service.

- The *Query Privileges Service* is utilized for the verification of privileges for the situation when a user (a client application or another resource of the distributed environment) is attempting to lookup on a (resource, service, method) triple. The critical method of this service is Check_Privileges(Token, R_Id, S_Id, M_Id, ParamValueList), which is used to ensure that a user playing a role (as identified by a token) is allowed to invoke a method (as identified by R_Id, S_Id, M_Id, ParamValueList) at the current time. The Query Privileges Service is tasked with ensuring that only allowable method invocations occur.
- The *User Role Service* is provided for the security officer to define and delete user roles. Recall that when defining a user role (see Background section), the name, lifetime, and classification must be provided, as is indicated in the method definition Create_New_Role.
- The *Constraint Service* is used in two different ways. The DefineTC and DefineSC methods are used by the security officer to specify time and signature constraints via the Security Policy Client tool, allowing the security officer to define the conditions under which the method can be invoked. The CheckTC and CheckSC methods are used by the Check_Privileges method of the Query Privileges service to allow these constraints to be dynamically verified at runtime when a user playing a role is attempting to invoke a method.
- The *Grant-Revoke Service* is used by the Security Policy Client for establishing privileges of each role. The security officer can grant each user role an entire resource (all of its services and their methods), a service (all of its methods), or individual methods, and for individual methods, these grants can be constrained based on time or signature constraints. Revocation methods of the Grant-Revoke Service remove the permissions.
- *Security Authorization Services* (Figure 4) are utilized to maintain profiles on the clients (for example, users, tools, software agents, and so forth) that are authorized and actively utilizing nonsecurity services, allowing a security officer to create users and authorize users to roles. There are three services:
- The *User Service* is the counterpart of the User-Role Service and is provided for the security officer to define and delete users. Recall that when defining a user (see Background section), the name, lifetime, and clearance must be provided, as is indicated in the method definition Create_New_User.
- The *Authorize Role Service* is the counterpart of the *Grant-Revoke Service* for the security officer to grant (for a limited time) and revoke a role to a user with the provision that a user may be granted multiple roles but must play only a single role when utilizing a client application.

Figure 3. The security policy services of USR

SECURITY POLICY SERVICES

```
Register Service
Register_Resource(R_I, TC, CLS);
Register_Service(R_Id, S_Id , TC, CLS);
Register_Method(R_Id, S_Id, M_Id, TC, CLS);
Register_Signature(R_Id, S_Id, M_Id, Signat);
UnRegister_Resource(R_Id);
UnRegister_Service(R_Id, S_Id);
UnRegister_Method(R_Id, S_Id, M_Id);
Unregister_Token(Token)

Query Privileges Service
Query_AvailResource();
Query_AvailMethod(R_Id);
Query_Method(Token, R_Id, S_Id, M_Id);
Check_Privileges(Token, R_Id,
               S_Id, M_Id, ParamValueList);

User Role Service
Create_New_Role(UR_Name, UR_Disc, UR_Id,
                                  LT, CLS);
Delete_Role(UR_Id);

Constraint Service
DefineTC(R_Id, S_Id, M_Id, TC);
DefineSC(R_Id, S_Id, M_Id, SC);
CheckTC(Token, R_Id, S_Id, M_ID);
CheckSC(Token, R_Id, S_Id, M_ID,
                        ParamValueList);

Grant-Revoke Service
Grant_Resource(UR_Id, R_Id);
Grant_Service(UR_Id, R_Id, S_Id);
Grant_Method(UR_Id, R_Id, S_Id, M_Id);
Grant_SC(UR_Id, R_Id, S_Id, M_Id, SC);
Grant_TC(UR_Id, R_Id, S_Id, M_Id, TC);
Revoke_Resource(UR_Id, R_Id);
Revoke_Service(UR_Id, R_Id, S_Id);
Revoke_Method(UR_Id, R_Id, S_Id, M_Id);
Revoke_SC(UR_Id, R_Id, S_Id, M_Id, SC);
Revoke_TC(UR_Id, R_Id, S_Id, M_Id, TC);
```

- The *Client Profile Service* is utilized by the security officer to monitor and manage the clients that have active sessions. The main method of interest for this chapter is Verify_UR(User_Id, UR_Id) which is used to verify if the user is allowed to play a specific role at the current time and is the counterpart of the Check_Privileges method of the Query Privileges Service.

The *Security Registration Services* (Figure 4) are utilized by clients at start-up for identity registration (client id, IP address, and user role), which allows a unique Token to be generated for each session of a client. These capabilities are all captured by the methods of the *Register Client Service.* Finally, there is the *Global Clock Resource (GCR*) and its associated method Get_Current_Time which is used by Security Policy Services to verify a TC when a client (via a UR) is attempting to invoke a method and Security Registration Services to obtain a common time, which is then used in the generation of a unique Token.

In order to illustrate the services given in Figures 3 and 4, the remainder of this section examines the processing required by a client that is joining the distributed environment

Figure 4. The security authorization and registration services of USR

SECURITY AUTHORIZATION SERVICES

```
User Service
Create_New_User(User_Name, User_Id, LT, CLR);
Delete_User(User_Id);

Authorize Role Service
Grant_Role(UR_Id, User_Id, TC);
Revoke_Role(UR_Id, User_Id, TC);

Client Profile Service
Verify_UR(User_Id, UR_Id);
Find_Client(User_Id);
Find_All_Clients();
```

SECURITY REGISTRATION SERVICES

```
Register Client Service
Create_Token(User_Id, UR_Id, Token);
Register_Client(User_Id, IP_Addr, UR_Id);
UnRegister_Client(User_Id, IP_Addr, UR_Id);
IsClient_Registered(Token);
Find_Client(User_Id, IP_Addr);
```

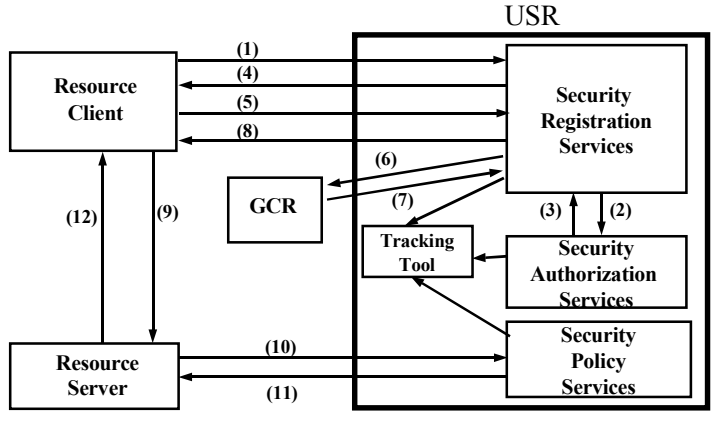
GLOBAL CLOCK RESOURCE

```
Get_Current_Time(): returns Time;
```

and attempting to access resources, which is given in Figure 5. In steps 1 to 4, the client is authenticated. In step 5, the client selects a role to play for the session. In steps 6 to 8, a token is generated and assigned to the user for the session via a name and password verification of the Unified Security Resource (USR). USR is a set of middleware security resources (Jini and CORBA) that manage all MAC and RBAC meta-data for users, user roles, and resources (Phillips et al., 2002a). The user chooses a role and registers via the RegisterClient method, which requests a global time from the GCR and returns a Token via CreateToken. In step 9, the client discovers the desired method from the lookup service (Jini or CORBA) and attempts to invoke the method with its parameters and the Token. In step 10, the resource uses the hasClientRight method (step 11) to check whether the user/client meets all of the MACC, time, and signature constraints required to invoke the method (step 12).

To further illustrate the process, consider an example using a university application, which is given in Figure 6. In the first step in the process, the user of the University Client must be authenticated to play a particular role. To do so, the University Client registers with USR via the Register_Client method (step 1), which must verify the user role (steps 2 and 3), and return a generated token via the Create_Token method (step 4). To generate

Figure 5. Client interactions and service invocations



1. Login (user_id, pass_wd)
2. isClientAuthorized()
3 getRoleList(user_id)
4 Return Role List
5. Register_Client (user_id, pass_wd, role)
6. Request Time from GRC
7. Return Current Global Time
8. If client time not expired, return Token
9. Make Resource Request
10. Check Authentication and Authorization Constraints via hasClientRight(role_id, R_id, S_id, m_id, Token, paramlist)
11. Return Results of Constraint Check
12. If Validated Request, Return Results to Client

the token in step 4, the Security Registration Service will interact with the Global Clock Resource. Since the GCR processes all requests sequentially, a unique time is always returned, and the token consisting of User-Id, UR-ID, IP address, and creation time is unique. A user with multiple sessions on the same machine with the same role has different tokens for each session as distinguished by the creation time.

Assuming that the registration and token generation was successful, the user can then attempt to utilize services from the University DB Resource. The University Client consults the lookup service for desired [resource, service, method] (step 5) which returns a proxy to RegisterCourse, allowing the method to be invoked (step 6) with the parameters Token, CSE230, and Martinez. The UnivDB Resource has two critical steps to perform before executing RegisterCourse. First, UnivDB Resource verifies that the Client has registered with the security services (steps 7 and 8). If this fails, a negative result is sent back via the RegisterCourse result (step 11). If this is successful, then the University DB Resource must perform a positive privilege check (privileges assigned by role) to verify if the user role can access the method limited by signature constraints and/or time constraints (both may be null). This is done in step 9 with two other method calls from Figure 3:

9a CheckTC(Token, UnivDB, Modification, RegisterCourse)

9b CheckSC(Token, UnivDB, Modification, RegisterCourse, [CSE230,Martinez]);

Figure 6. Service invocations and processing for university application

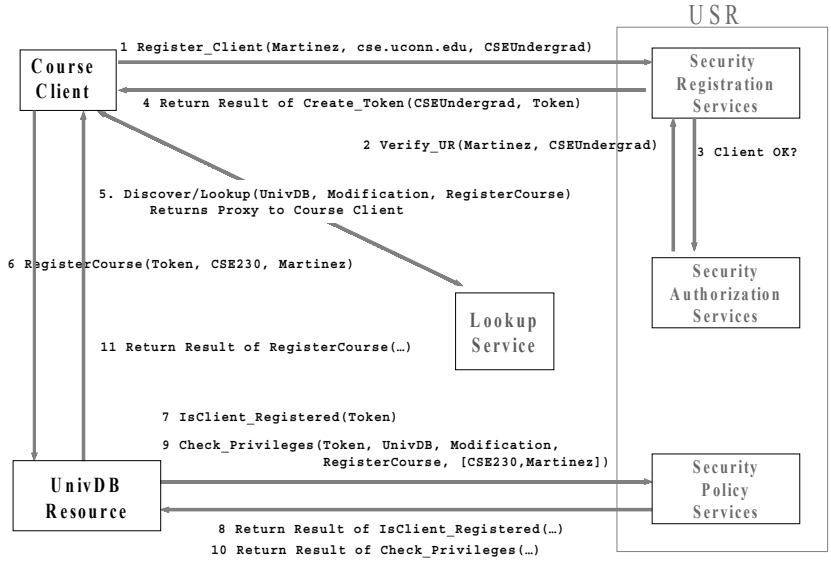
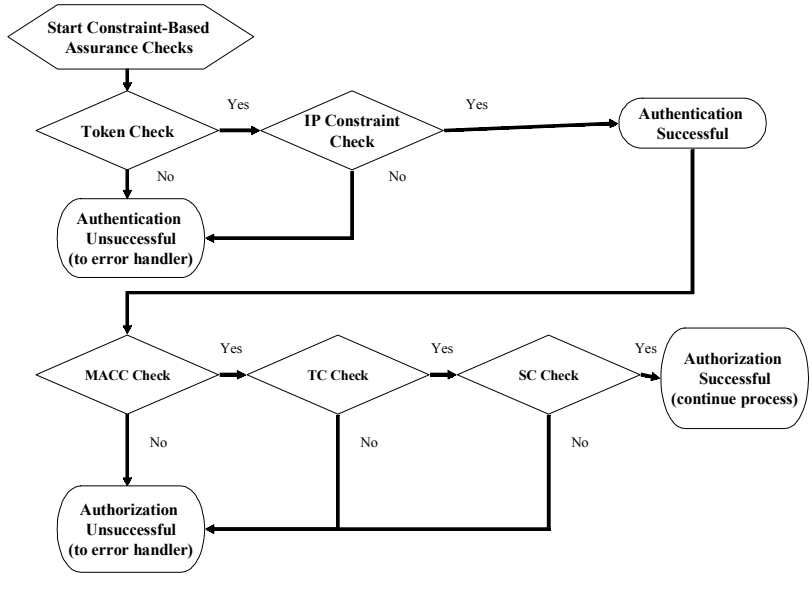


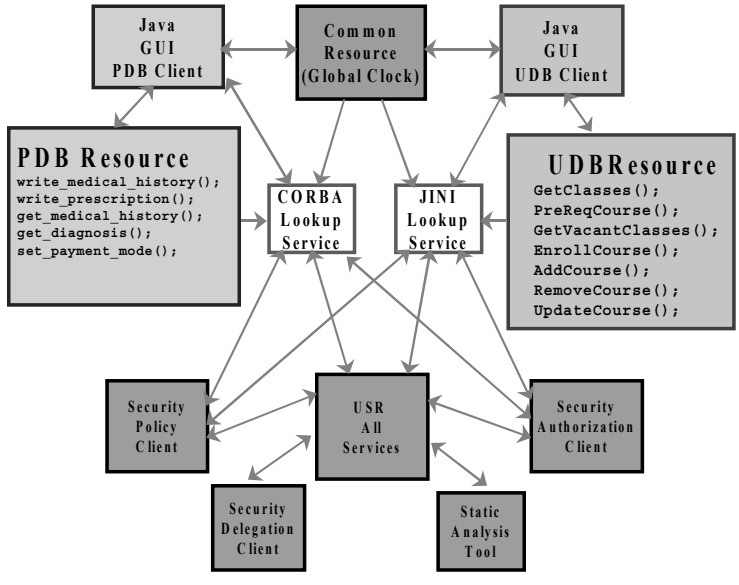
Figure 7. Assurance checks performed by security services



The CheckTC method interacts with GCR to verify that the current time is within the limits of the time constraint (if present). If the privilege check is successful (step 10), then the method executes as called, and the result (registering Martinez for CSE230) is returned as a success in step 11. Otherwise, the result (step 10) denies the registration via step 11. Note that the signature constraint is verified in two phases. The parameters constraint (if present) must be checked prior to method invocation, while the return-type constraint (if present) must be checked after execution and before the result has been returned.

In summary, in order to obtain access to a service or method, there are a series of checks as shown in Figure 7: *Client Authentication* with *Token Check* on the validity of the token and *IP Constraint Check* to verify if the user is logged on from a permitted location and *Client Authorization* with a *MACC Check* to verify that the user/client has the required CLR to access the method, a *TC Check* to verify if the access is within the allowable time period, and a *SC Check* to verify that the invocation satisfies the signature constraint. Collectively, all of the checks illustrated in Figures 5 and 6 provide runtime assurance of the RBAC/MAC policy as a client (with a UR and CLR) invokes methods (with CLS) of resources.

Figure 8. Prototype security and resource architecture



Prototyping: Administrative/Management Tools

This section reviews the prototyping for our service-based security model and enforcement framework. As shown in Figure 8, we have designed and implemented the entire security framework as given in Figure 1, which includes the USR and administrative/management tools, capable of interacting with either CORBA or Jini as the middleware. We have also prototyped a University DB resource and client and a hospital application where a client can access information in a Patient DB (Figure 8). The hospital portion of Figure 8 (left side) interacts with CORBA as the middleware; the university portion (right side) uses Jini. Note that since the lookup service is transparent to the user, a client could be constructed to use CORBA and/or Jini depending on which one is active in the network. From a technology perspective, the university application in Figure 8 (right side) is realized using Java 1.3, Jini 1.1, Windows NT 4.0 and Linux, and Oracle 8.1.7. The hospital application (left side) uses the same technologies except for Visibroker 4.5 for Java as middleware. Both resources (Patient DB and University DB) operate in a single environment using the shared USR and are designed to allow them to register their services with CORBA, Jini, or both. The University DB Java Client allows students to

Figure 9. Security policy client - defining a role

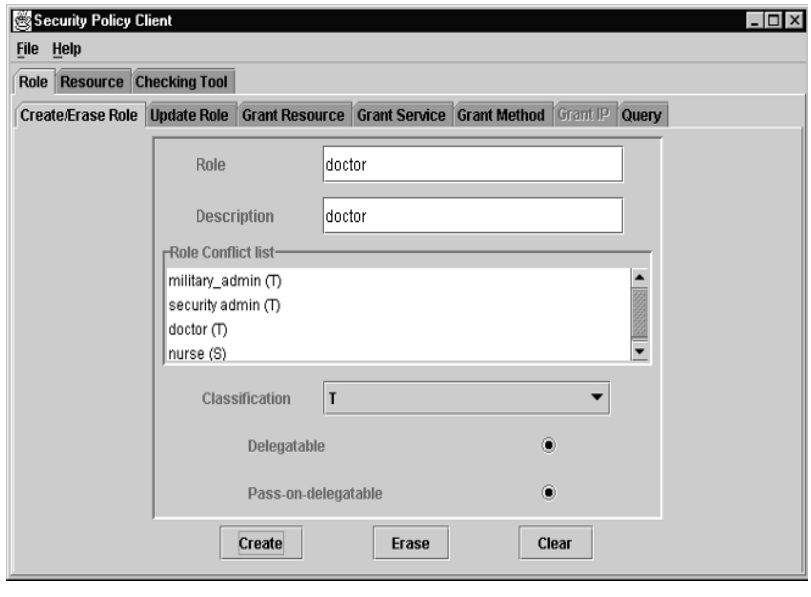
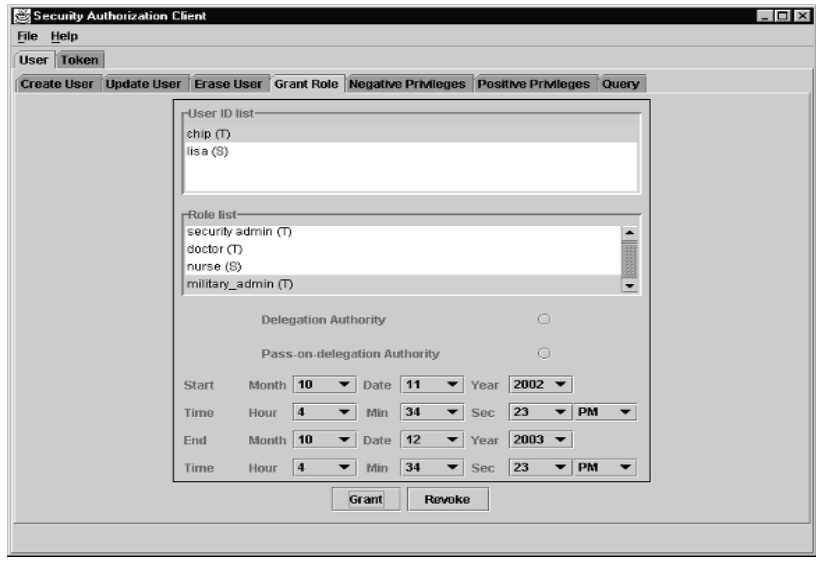


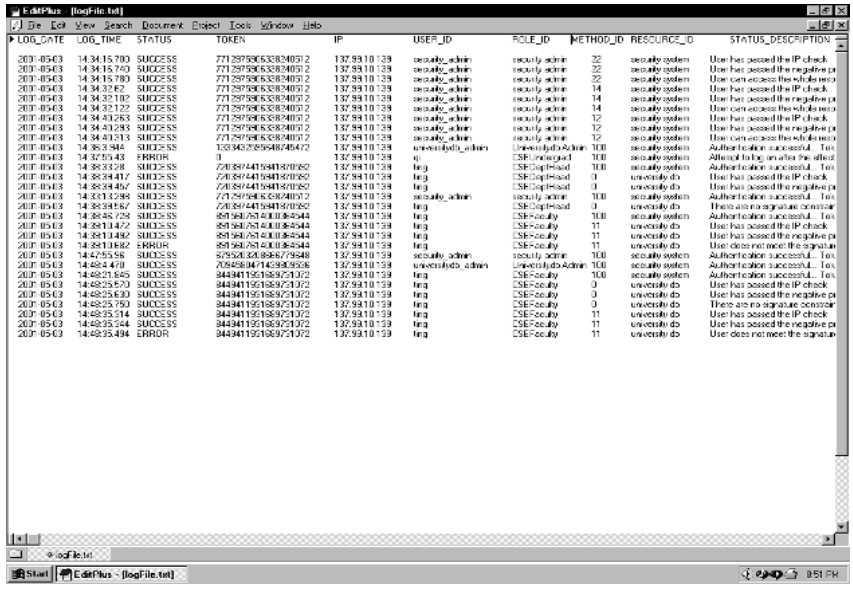
Figure 10. Security Authorization Client - Granting a Role



query course information and enroll in classes and faculty to query and modify the class schedule. The Patient DB Java Client supports similar capabilities in a medical domain.

In addition, the prototyping environment supports administrative and management tools, as shown in the bottom portion of Figure 1. The *Security Policy Client (SPC)*, shown in Figure 9, can be used to define and remove roles and to grant and revoke privileges (that is, CLR/CLS, time constraints, resources, services, methods, and/or signature constraints). The security officer can also inspect and monitor security via the tracking capabilities of SPC. Also, SPC is used to establish whether a role is delegatable. In Figure 9, the tab for defining a user role, assigning a classification, and determining the delegation status, is shown. The *Security Authorization Client (SAC)*, shown in Figure 10, supports authorization of role(s) to users. A user may hold more than one role but can only act in one role at a time. SAC provides the security officer with the ability to create a new User as discussed in the previous section. SAC is also used to authorize role delegation. In Figure 10, a user is being granted access to role for a specific time period. Not shown is the *Security Delegation Client (SDC)* (see Liebrand et al., 2003; Uconn, 2003), which is intended to handle the delegation of responsibilities from user to user.

Figure 11. Security Analysis Tool - Access History



In addition to the definitional capabilities of SPC and SAC, Figure 11 is an example of one output of the *Security Analysis Tool (SAT)* which is used to dynamically track all client activity, including log-ons and method invocations. SAT is intended as a means to allow security personnel to watch and track behavior within the system, providing a tracking capability for all authorized and attempted unauthorized access. SAT utilizes many of the different services in Figures 3 and 4.

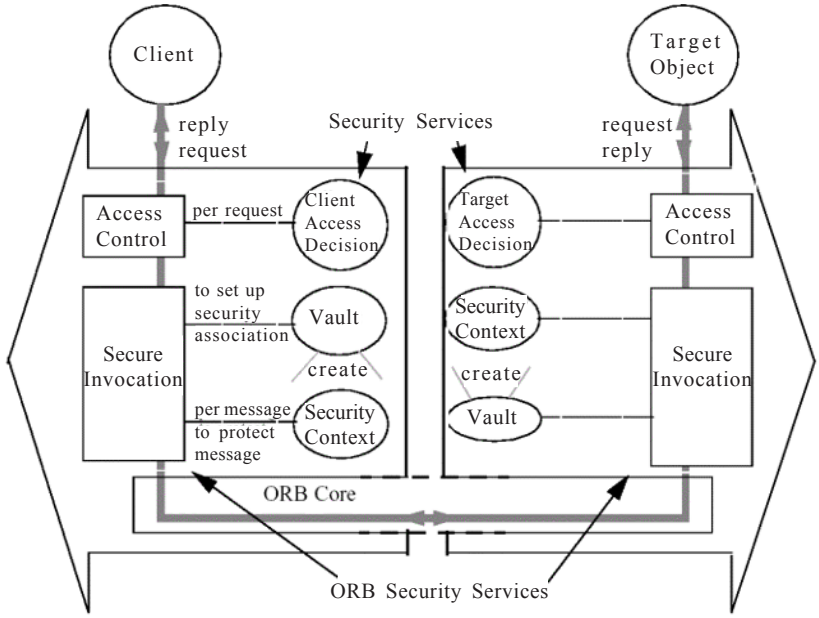
Future Trends

To complement our own approach to middleware security, it is relevant to examine the emerging trends for support of security in a middleware setting. Modern middleware platforms like CORBA (Object Management Group, 2002), .NET (Microsoft Corporation, 2003a), and J2EE (Sun Microsystems, 2003) have begun to offer security capabilities. In assessing these three approaches (Demurjian et al., 2004), the major difference between the support for security in CORBA (as opposed to its realization in an actual CORBA product, for example, Visibroker) and security in .NET/J2EE is that the CORBA security

specification is a *meta-model*. As a meta-model, the CORBA security specification generalizes many security models and associated security principles (wide variety of security capabilities at the model level — RBAC, MAC, encryption, and so forth) with language independence (not tied to Java, C++, .NET, and so forth). .NET and J2EE provide actual security capabilities via their respective runtime environments and APIs, which provide security functionality. The remainder of this section explores the security capabilities of CORBA, .NET, and J2EE.

The CORBA Security Service Specification (Object Management Group, 2002a) focuses on confidentiality (limiting access to authorized individuals/programs), integrity (limiting modifications to authorized users), accountability (requiring users to be responsible for their actions), and availability. These aspects are part of the CORBA security reference model, given in Figure 12. In this model, the access control process verifies a subject's permissions *(via privilege attributes)* against the target objects which are managed via *control attributes (grouped as domains)* and operations (grouped as *rights*). *Privilege attributes* are associated with the user *(principal)*, and the permissions tracked for each principal are: identity (user id), role(s), group(s) that the principal belongs to, security clearance (for example, secret, classified, and so forth), and target objects and operations to which has been granted access. From a complementary perspective, *control attributes* track the security privileges for each target, for example, an access control list entry for a target object would track the security characteristics of the object (for example, security classification), the *rights* of a target object (that is, the

Figure 12. The CORBA security model (Object Management Group, 2002b)

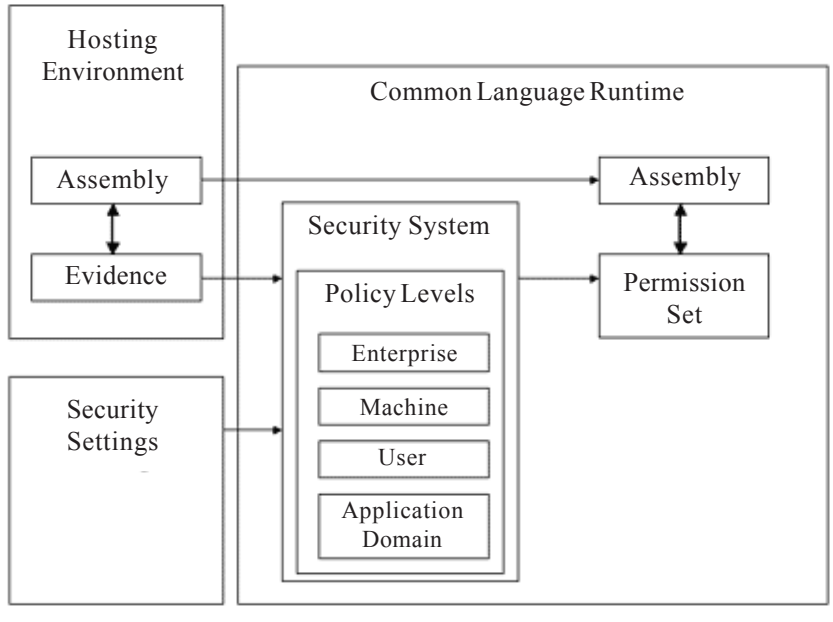


set of operations that are available for assignment to each principal), and the principals who have been authorized.

In order to define privileges for principals and target objects, a *security policy domain* is used to represent the scope over which each security policy is enforced, assuming that an organization may have multiple policies. A security policy domain permits the definition of security requirements for a group of target objects, allowing this group to be managed as a whole, thereby reducing the needed administrative effort. A *policy domain hierarchy* allows a security administrator to design a hierarchy of policy domains and then delegate subsets of the hierarchy (subdomain) to different individuals. As a meta-model, the CORBA security specification is robust enough to realize RBAC, MAC, or any other security model by customizing the concepts of principal, privilege attributes, target objects, control attributes, and policy domains to suit the desired security model nomenclature.

This structural model of security of .NET (Microsoft Corporation, 2003a), as represented in Figure 13, consists of the Common Language Runtime (CLR), the Hosting Environment, and the Security Settings. For the Hosting Environment to execute an application, it must provide the code (via assembly — compiler generated code) and its identity (via evidence — *proof* that is supplied regarding identity) in its interactions with CLR. CLR contains the Security System, which realizes the security policy at enterprise, machine, user, and application domain levels. For an actual application, the different parameters related to

Figure 13. .NET security structural model (Microsoft Corporation, 2003b)



security must be set within the Security System, as shown by the input from the Security Settings box in Figure 13, to establish the security at one or more policy levels. For execution to occur within CLR, the assembly is used to identify the required permission set (for example, allowances given to a piece of code to execute a certain method) and be provided with evidence from the Host to the Security System.

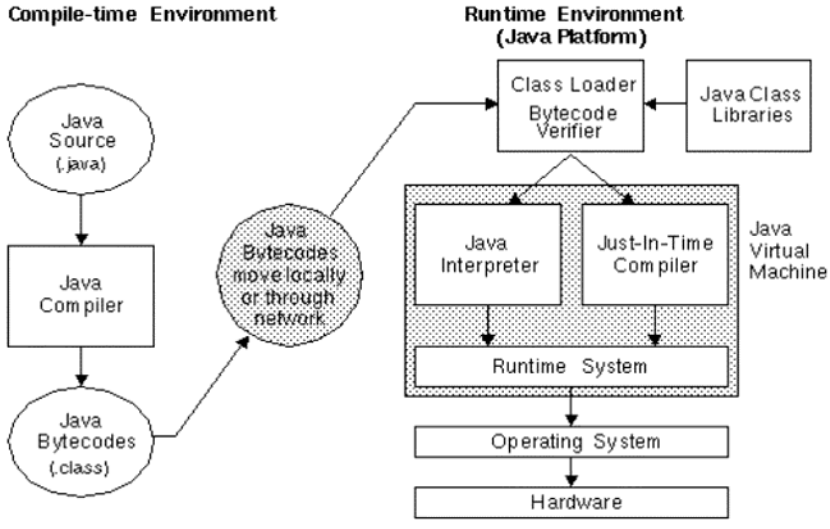
Code-based access control, a part of CLR's security, dictates the situations where access by a code segment to a resource is permitted (prevented). The determination of what a piece of code is allowed to do is decided by *evidence-based security*, *permissions*, and a *security policy*. During execution, the CLR reviews evidence of an assembly, determines an identity for the assembly, and looks up and grants permissions based on the security policy for that assembly identity (Open Web Application Security Project, 2002). *Evidence-based security* determines the origin(s) of an assembly. At runtime, the CLR examines the meta-data of an assembly for the origin of the code, the creator of the assembly, and the URL and zone (for example, Internet, LAN, local machine, and so forth) of the assembly.

The successful verification of evidence leads to the *permissions* of code and code segments, which is the ability to execute a certain method or access a certain resource. An assembly will request permissions to execute, and these requests are answered at runtime by the CLR, assuming that the assembly has provided *apropos* evidence. If not, CLR throws a security exception, and an assembly's request is denied. Since numerous different permissions can be requested, permissions are grouped into sets. Permissions and permission sets in .NET are similar to privilege/control attributes and domains in CORBA, respectively. As such, it is possible to establish permissions for MAC classification levels for code and resources with the access permitted or denied based on the domination of the code's classification over the resource's classification.

Lastly, the grouping of assemblies based on different criteria establishes different *security policies* for different code groupings (Microsoft 2003a, Open Web Application Security Project, 2002). In .NET, there are three different security policies that are supported: enterprise level for a cohesive and comprehensive policy for the entire enterprise, machine level for different policies for different machines, and user level to capture individual responsibilities. The .NET framework provides the means to organize security policy groups of assemblies into hierarchical categories based on the identity that the CLR determines from the evidence. Once related assemblies have been grouped and categorized, the actual security policy can be specified as permissions for all assemblies in a group. RBAC in .NET extends the policies and permissions concepts of code-based access control to apply to a user or role. .NET uses role-based security to *authenticate* an identity and to pass on that identity to resources, thereby *authorizing* the users playing roles access to resources according to policies and permissions.

Security in the Java 2 Enterprise Edition (J2EE) (Sun Microsystems, 2003) focuses on its ability to keep code, data, and systems safe from inadvertent or malicious errors. In Figure 14, the compilation of Java code creates bytecode, whose execution involves the class loader (with bytecode verifier), the Java class libraries (APIs), and the Java virtual machine (JVM). The JVM manages memory by dynamically allocating different areas for use by different programs, isolating executing code, and performing runtime checks. The block labeled Runtime System, as shown in Figure 14, contains the *Security Manager*,

Figure 14. The Java 2 platform - compile and execute



Access Controller, and other features that all interact to maintain security of executing code. Security considerations in J2EE are important for both applications and applets, but applets are of particular concern for security, since they represent remote code that is brought in and executed on a local machine. To control applet behavior, Java uses a sandbox, which forces downloaded applets to run in a confined portion of the system and allows the software engineer to customize a security policy. The *Security Manager* enforces the boundaries around the sandbox by implementing and imposing the security policy for applications. All classes in Java must ask the security manager for permission to perform certain operations. Java only has two security policy levels, one for the executing machine and one for the user. Each level can expand or restrict on all of the permissions of another level, and there can be multiple policy files at each level.

Permissions in Java are determined by the security policy at runtime and are granted by the security policy based on evidence. The evidence that Java looks for is a publisher signature and a location origin. Permissions are also grouped into protection domains (similar to security policy domains in CORBA and to security policy files in .NET) and associated with groups of classes in Java in much the same way they are grouped into

permission sets and associated with code groups in .NET. However, in Java, MAC is not automatic; it requires programmatic effort by the software engineer.

In support of RBAC, J2EE uses the Java Authentication and Authorization Service (JAAS), which implements a Java version of the Pluggable Authentication Module framework. Using JAAS, software engineers are allowed to modify and then plug in domain/application-specific authentication modules (DevX Enterprise Zone, 2002). JAAS currently supports authentication methods including UNIX, JNDI, and Kerberos, akin to OS level security. JAAS can only provide limited impersonation authentication because the user identity is different for the application and OS levels. User access checking can be done both declaratively and imperatively within different components ofJ2EE.

Concluding Remarks

In this chapter, we have explored a middleware services solution for a unified RBAC/MAC security model and enforcement framework for a distributed environment (Liebrand et al., 2003; Phillips et al., 2002a; Phillips et al., 2002b; Phillips et al., 2003a; Phillips et al., 2003b) where clients and resources interact via services implemented in Jini and Visibroker. In the approach, client interactions to resource APIs are controlled on a role-by-role basis, constrained by security level, time, and data values. This work is interesting since it demonstrates that service-based software can be an extremely effective tool to easily and seamlessly integrate complex security capabilities. In addition, we have examined the future trends of the support by security in middleware by exploring the security capabilities and features of CORBA, .NET, and J2EE. Overall, we believe that the material presented in this chapter can be used as a basis to understand the design and integration of security into distributed applications via a service-based approach.

References

Arnold, K., et al. (1999). *The JINI specification*. Addison-Wesley.

Demurjian, S., et al. (2004). Concepts and capabilities of middleware security. In Q. Mahmoud (Ed.), *Middleware for communications.* John Wiley & Sons.

Demurjian, S., et al. (2001). A user role-based security model for a distributed environment. In J. Therrien (Ed.), *Research advances in database and information systems security*. Kluwer.

DevX Enterprise Zone. (2002). Software engineers put .NET and Enterprise Java Security to the test. Retrieved August 19, 2004, from *http://www.devx.com/enterprise/articles/dotnetvsjava/GK0202-1.asp*

Liebrand, M., et al. (2003). Role delegation for a resource-based security model. In E. Gudes & S. Shenoi (Eds.), *Data and applications security: Developments and directions II*. Kluwer.

Microsoft Corporation. (1995). *The component object model (technical overview)*. Microsoft Press.

Microsoft Corporation. (2003a). Microsoft DN, Microsoft .NET security. Retrieved August 19, 2004, from *http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001369*

Microsoft Corporation. (2003b) Microsoft TechNet. Security in the Microsoft .NET framework. Retrieved August 19, 2004, from *http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/net/evaluate/fsnetsec.asp*

Object Management Group. (2002a). Common object request broker architecture: Core specification – version 3.0.2. Retrieved August 19, 2004, from *http://www.omg.org*

Object Management Group. (2002b). Security Service Specification - Version 1.8, March 2002, Figure 2-47, page 2.65. *http://www.omg.org*

Open Software Foundation. (1994). OSF DCE application development guide - revision 1.0.

Open Web Application Security Project. (2002). J2EE and .NET security. Retrieved August 19, 2004, from *http://www.owasp.org/downloads/J2EEandDotNetsecurityByGerMulcahy.pdf*

Phillips, C., et al. (2002a). *Security engineering for roles and resources in a distributed environment.* Proceedings of the 3rd Annual ISSEA Conference.

Phillips, C., et al. (2002b). *Towards information assurance in dynamic coalitions.* Proceedings of the 2002 IEEE Information Assurance Workshop.

Phillips, C., et al. (2003a). *Security assurance for an RBAC/MAC security model.* Proceedings of the 2003 IEEE Information Assurance Workshop.

Phillips, C., et al. (2003b). *Assurance guarantees for an RBAC/MAC security model.* Proceedings of the 17th IFIP 2002 11.3 WG Conference.

Riordan, R. (2002). *Microsoft ADO.NET step by step.* Microsoft Press.

Roman, E. (1999). *Mastering Enterprise JavaBeans and the Java 2 Platform, enterprise edition.* John Wiley & Sons.

Rosenberry, W., Kenney, D., & Fischer, G. (1992). *Understanding DCE.* O'Reilly & Associates.

Sceppa, D. (2002). *Microsoft ADO.NET (core reference)*. Microsoft Press.

Sun Microsystems. (2003). J2EE security model. Java 2 platform security. Retrieved August 19, 2004, from *http://java.sun.com/j2se/1.4.1/docs/guide/security/spec/security-spec.doc.html*

UConn. (2003). Distributed security. Retrieved August 19, 2004, from *http://www.engr.uconn.edu/~steve/DSEC/dsec.html*

Valesky, T. (1999). *Enterprise JavaBeans: Developing component-based distributed applications*. Addison-Wesley.

Vinoski, S. (1997). CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine, 14*(2).

Waldo, J. (1999). The JINI architecture for network-centric computing. *Communications of the ACM, 42*(7).

Yang Z., & Duddy, K. (1996). CORBA: A platform for distributed object computing. *ACM Operating Systems Review, 30*(2).

Section V

Service-Orientation in Practice

Chapter XVI

Engineering a Service-Oriented Architecture in E-Government

Marijn Janssen
Delft University of Technology, The Netherlands

Abstract

Service-oriented enterprise architectures have gained considerable attention of politicians and public servants as a solution for designing new applications and leveraging investments in legacy systems. Service-oriented architectures can help to share data and functionality among information systems and provide the flexibility to include existing legacy systems, which cannot be replaced easily and otherwise restrict further development. In this chapter, the design of a service-oriented architecture in public administration is explored. A case study is conducted at the Ministry of Justice, and a service-oriented architecture is designed, implemented, and evaluated. The architecture is evaluated based on a number of quality requirements. This case study shows the feasibility to replace functionality formerly offered by legacy systems and shows limitations of current technology. This chapter should lead to a greater understanding of the concept of service-oriented architectures in e-government.

Introduction

Electronic government (e-government) is emerging as the result of technological developments in the field of Information and Communication Technology (ICT) and is reshaping relationships (Chen, 2002). Government organizations are becoming increasingly aware of the opportunities provided by service-oriented architectures. *ICT-Architecture* is the description of the set of components and the relationships between them (Armour, Kaisler & Liu, 1999). The basic idea is to break a large and complex system down into relative simple parts. The parts can be designed individually, and a new system can be constructed by (re)using the relative simple parts as configurable, modular services. The service-oriented paradigm focuses on building information systems by discovering, matching, and integrating predeveloped components as services. Generally, a service functions independently of other components and communicates using well-defined standardized interfaces. The manageability increases as large modular services can be constructed from smaller ones. In essence, a complex problem is split up into smaller problems, which can be solved independently. Openness and flexibility is created as new services can be added and removed from the architecture, and each single component can be replaced by another component without affecting the others. Service-oriented modular architectures can leverage investments in legacy systems running the enterprise's key business-critical applications (Arsanjani, 2002).

The service-oriented paradigm offers many benefits to enterprises, and the creation of a class of enterprise services allows us to create services that are modular, accessible, well-described, implementation-independent, and interoperable (Fremantle, Weerawarana & Khalaf, 2002). Modular services can be found, described, discovered, and integrated using Web services technology. Service-oriented paradigms are becoming more important in today's design of information systems. Service-oriented business integration enables the on-demand composition of new business processes using already existing services possibly provided by other parties. In service-oriented applications, services are configured to meet a specific set of requirements at a certain point in time, executed and then disengaged. Services only *exist* during execution; components provide services.

A few Web services protocols have become the *de facto* standard by the Dutch government. Many architecture departments have adopted HTTP, XML, XSLT, XML Schemas, SOAP, UDDI, and WSDL as standards for designing service-oriented architectures. Although the basic protocols have been set, and the concept of service-oriented architectures has great promises, their application still stays far behind in the public sector. Most government organizations are relatively slow in adapting service-oriented architectures as they lack sufficient insight into the real pros and cons of such an approach (Fan, Stallaert & Whinston, 2000).

The objective of this chapter is to explore the concept of a service-oriented architecture in e-government. In the following section, a case study will be introduced, and the quality requirements on a service-oriented architecture will be discussed. In the section thereafter, a service-oriented architecture levering investments in legacy systems and where some functionality of legacy systems has been replaced using Web services technology is discussed. Using interviews, the quality requirements are evaluated, and the service-

oriented architecture is discussed. Based on this discussion, conclusions are drawn in the last section.

Introduction to the Case Study

The Ministry of Justice in the Netherlands has more than 14,000 employees working in the colleges of judges and public prosecutors. There are 19 county courts, five courts, and one supreme court and three types of law: civil, public administration, and criminal. Between courts and often for each type of law within one court, different kinds of information systems are used. This ministry is continuously developing new applications and replacing existing systems. In the past, development projects, control, and maintenance of information systems were not coordinated. Each information system was designed using different knowledge, often coming from external consultants. This has resulted in information systems residing on the MS Windows, VMS, and UNIX platforms, written in languages like PL/SQL, C++, and Java using applications from Oracle, Microsoft, and other vendors. All by all, this resulted in 22 heterogeneous types of information systems having overlap in functionality and each having its own control and maintenance team. In total, more than 250 FTE are involved in maintaining and controlling the systems.

The Ministry of Justice wants to introduce a more open and flexible service-oriented architecture. The interest in a service-oriented architecture is coming from the following aspects:

1. *Modularity*: Components are reusable, and new components and business processes can be composed out of existing components. Legacy systems are also viewed as modules accessible as a service.
2. *Implementation-independent interfaces:* Interfaces can be described using WSDL independent of the implementation. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL supports the description of multiple implementations.
3. *Use of registry*: Services descriptions can be registered in a directory service based on UDDI. Users and information system can discover services and integrate them into their business processes.

An architecture is a high-level map of the information requirements of an organization. It is a personnel-, organization-, and technology-independent profile of the major information categories within an enterprise (Branchau, Schuster & March, 1981). Over time, the courts, users, and ICT Department have agreed on over a number of quality attributes, described in Table 1, which can be viewed as requirements on the architecture.

Two types of attributes can be distinguished: user attributes, which are quality requirements taken from the user perspective and design attributes, which determine the adaptability of the architecture from the developers point of view. Only for the performance and availability attributes are quantitative norms available. The response time for users may not exceed 0.5 seconds. The availability should be 99% during daytime (8.30-18.00) and 99% during special time windows to accommodate crime investigations.

In Figure 1, the desired growing strategy of the service-oriented architecture is mapped out in the time. Messages are the most elementary part of the architecture. A message can be a simple signal, such as an acknowledgment of delivery, or it can have a more complex nature and contain whole documents. During the first step, messages should be converted to SOAP/XML-based protocols. In this way, interoperability between systems should be guaranteed. In the following step, a number of services should be implemented and made available within the Ministry, such as document generating, scheduling of appointments, document retrieval, and so forth. Web Services Description Languages (WSDL) can be used to describe the services.

In the following phase, a services catalog based on the Universal Discovery and Integration (UDDI) standard should be used to store, find, and integrate services. Initially, the UDDI registry is only for use within the Ministry of Justice, and the services will not be provided to other parties. This has advantages for maintainability, updating, monitoring, and forecasting use. Transactions are composed of messages or other transactions.

A business process orchestrates transactions into a sequence of activities in times. Although there is still disagreement over the selection of orchestration languages, the Business Process Execution Language for Web services (BPEL4WS) was selected as this seemed to be the most promising language at this moment (Fremantle et al., 2002; Thatte,

Table 1. Quality requirements on architecture

Quality attributes	**Description**
User attributes	
Performance	The time required to respond to stimuli (events)
Availability	The proportion of time the system is up and running
Personalization	The ability to personalize systems output to the user wishes
Security	The ability to resist unauthorized attempts at usage and denial of service while still providing services to legitimate users
Time-to-market	When new laws are introduced, a system has to comply to it prior to its activation
Design attributes	
Modifiability	The ability to make changes quickly and cost effectively
Portability	The ability of the system to run under various computing environments
Reusability	The ability to reuse some or all of the components of the system
Integratability	The ability to make the separately developed components of the system work together
Testability	The probability that a system will fail on the next use. The components input needs to be controlled, and the internal state and outputs need to be observed
Use of legacy system	New systems must integrate with existing systems which cannot be replaced easily

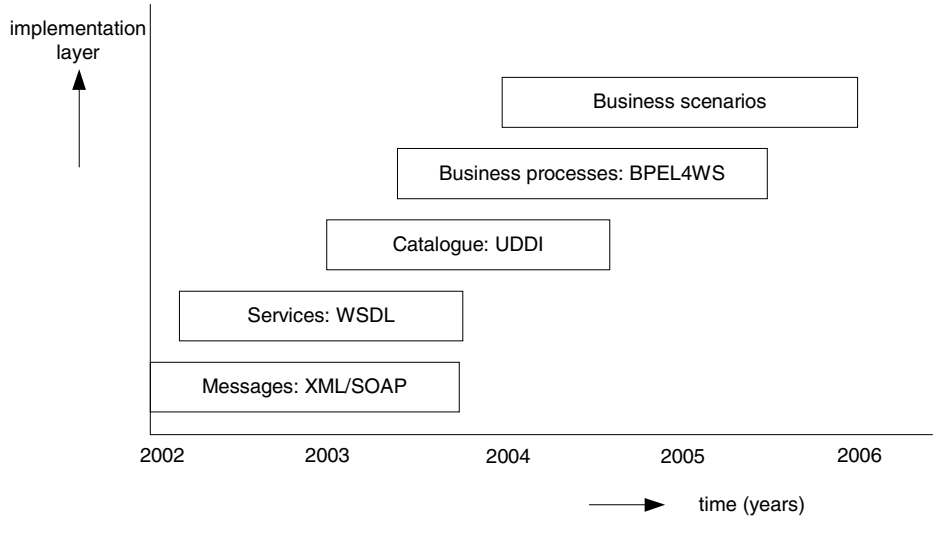
2002). In the last phase toward a complete SOA, business scenarios should be described, that is, the context in which business processes can take place and the dependencies between business processes. This includes partner profiles, describing the capabilities of each partner, contracts, rights, responsibilities, obligations, and so on.

To investigate the feasibility and value of a service-oriented architecture to accommodate the requirements of the Ministry of Justice and to explore the feasibility of the first phases a pilot was started. Initially, the pilot was limited to investigate the possibilities of messages (XML/SOAP), services (WSDL), and service registry (UDDI) to build a service-oriented architecture that could include the legacy systems currently in use. When this part showed to be feasible, a succeeding pilot project might be initiated to test and evaluate orchestration and business scenario within an SOA.

Engineering a Service-Oriented Architecture

Currently, a large project called GPS is initiated to replace a large and complex legacy system (COMPASS) in criminal law using innovative technologies. The names refer to tools supporting the finding of the right track. GPS consists of several projects: one is a pilot project aimed at verifying the concepts of modular services discussed in the preceding section and to explore the opportunities and limitations coming from the application of readily available technology. Document generation was chosen as modular functionality to be made accessible using services, as all existing information

Figure 1. Growing strategy



systems within the Ministry of Justice have to generate and print documents in some way or another.

Functional Requirements

The current functionality of the information systems and the organizational characteristics were taken as a starting point for the requirement elicitation. Further requirement elicitation was conducted using interviews. The general requirements are summarized hereafter.

- Documents consist of static and dynamic parts. The static data is predefined and makes up the body of the document containing the position of dynamic data, layout, law text, and logos. Elements of the static part can be filled with dynamic data like name, address and so forth.
- Dynamic data should be extracted from existing and potentially new databases.
- Documents like summons that seem to be similar on the first sight are different for each court. As there are in total 25 courts (one supreme court, five high courts, and 19 courts), content managers in a court should be able to update the content of the static part of a document.
- Roughly speaking, two types of documents should be created. The first type is incidentally created by users and is immediately needed. The other type of document is created automatically by the system based on the progress and status of cases. Often, these documents are generated at night and concern large volumes. At an average court, 150 summons are created each night.
- It should be able to generate documents that are stored in a document management system that cannot accidentally be changed, as they might be needed for evidence. It should also be possible to generate documents that can be changed by court employees before printing.
- There should be one single point where the document generator is developed, controlled, and maintained in a technical sense.
- A prerequisite is the use of Web services technology.

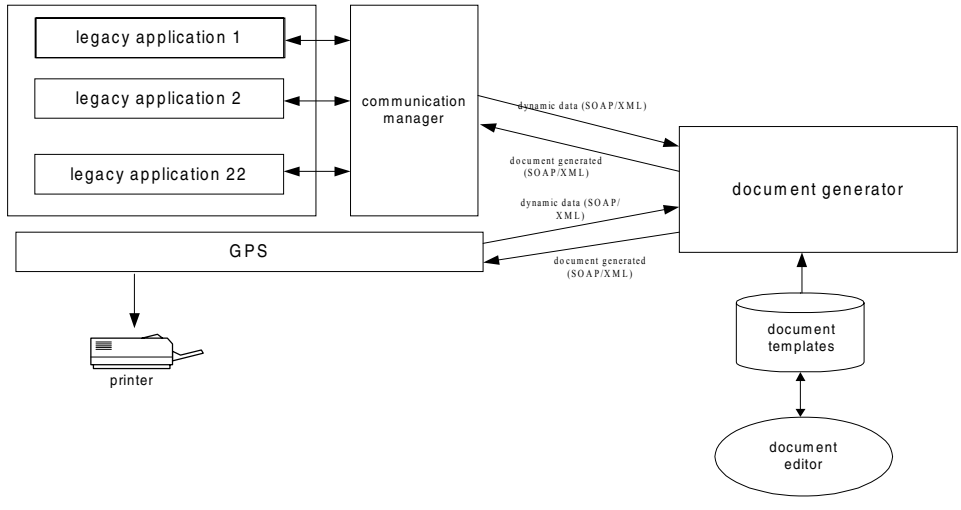
The static part of a document is implemented as a template. Local content managers in each court can create, edit, and save static document templates using a document editor in MS Word. For maintenance purposes, the templates are stored and retrieved at a central place and not local at a content manager's computer. The advantages of this construction are that a backup can be automatically made after each change and that in case of conversion of document formats, all document templates can be converted at once. The document generator should be able to generate RTF for use within a word processor and PDF documents to ensure that the content cannot easily be changed.

Implementation of the Document Generation Service

The service-oriented architecture for document generation is schematically shown in Figure 2. The communication manager mediates communication between legacy systems and the document generation service. The new information system can communicate directly with the document generator by sending an XML/SOAP message. Legacy systems are made accessible as services using adapters, in this way, concealing the complexities of the interface. Legacy systems, GPS, and the communication manager communicate with the document generator using XML message over SOAP and HTTP. XML (eXtensible Markup Language) is a language for describing hierarchical structured documents using tags. The SOAP (Simple Object Access Protocol) consists of an envelope/body structure and defines the way applications can request and deliver data using XML. The HTTP (Hypertext Transfer Protocol) is a simple request-respond protocol for communication between a client and server.

The legacy systems communicate with the document generator using the *communication manager*. The communication manager is a middleware application using message queuing based on Oracle technology. When a message is submitted, the message is placed into a persistent queue. In the following step, the communication manager gets the message from the incoming queue, translates the messages from a legacy system into an XML format, and the message is placed into the outgoing queue. The communication manager also performs validity checks on data structures and completeness. A Java client, which is also part of the communication manager, takes the message out of the outgoing queue and submits the message based on XML/SOAP to the document generator. The document generator is only able to read XML-based messages. In the opposite direction, the communication manager translates XML messages coming from

Figure 2. Overview of the service-oriented architecture



the document generator into a format readable by the legacy application. In this way, communication with components not supporting HTTP and XML message formats becomes possible.

Service descriptions are necessary for using the document generator in an open market situation. WSDL is used to describe the properties of services and to make comparison with other descriptions possible. A registry based on UDDI can be used to discover and integrate services.

In Figure 3, screen shots of the prototype are shown to demonstrate the document generation process. The screen shot on the left side shows a SOAP envelope loaded within a Web browser. Dynamic data is packaged within the SOAP envelope. The SOAP envelope contains information about the type of document that needs to be generated (RTF, Rich Text Format and PDF, Portable Document Format) and the static template and styles that should be used. The SOAP body contains the dynamic data that should be inserted into the document and extracted from a database. After a document is generated, the results are sent back in a SOAP envelope shown in the middle of Figure 3. The SOAP envelope contains information about possible communication failures, the type of document returned (RTF, PDF), and the body contains the document, in our example, an RTF file. The screen shot on the right side of Figure 3 shows the generated RTF document after the generated document was loaded into a word processor.

Document Generation Process

For the new information systems, GPS, the process needed for generating documents like summons is shown in Figure 4. The event-manager takes care of the orchestration of invoking Web services. A process is triggered by (1) a client (human) request to generate some kind of document, which often requires the immediate generation of a document, or (2) a batch process, which is often started at night as there is no urgent need for generating the documents immediately. In the case of a batch process, the request for generating a document is generated by the workflow application. Based on events like

Figure 3. Example of the process of generating a summon

dynamic input data for document generator (SOAP/XML)

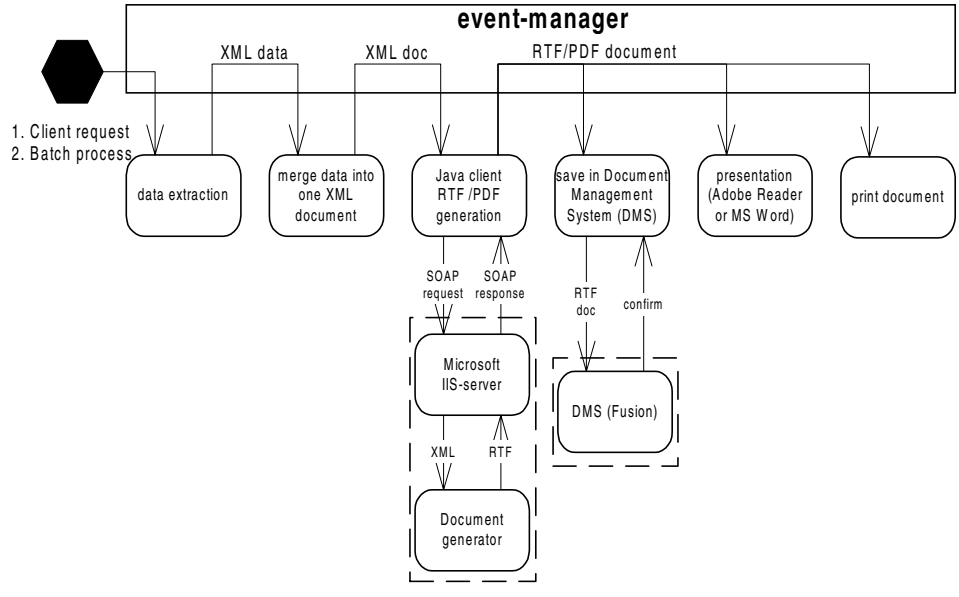
results document generation (SOAP/XML)

Document generated (RTF)

a public hearing or trial, the involved persons need to be sent a summons. To generate a document, first, data need to be extracted from a database and wrapped up in the body of an XML message. Often the data extraction requires more than one query; therefore, multiple queries need to be merged into the body of one single XML message and the type of document that needs to be generated, and the document format (RTF/PDF) needs to be added in the envelope of the message.

Using a simple client written in the Java language, a SOAP request is submitted to a Web server, which unpacks the SOAP message and invokes the document generator by providing information about the template to be used, the type of document to be generated, and the dynamic data extracted from the database. An RTF is generated and wrapped up as a SOAP message by the Web server, which submits this message to the Java client. The event-manager can store the document, type of trigger, generation data, and other data in a Document Management System (DMS). If a human client request triggers, the document generation process, the document can either be presented in Adobe Reader (if a PDF is generated), or MS Word (if an RTF is generated). The document is printed if a batch process initiates the document generation process. Often, printing is performed at night, and printed documents are automatically inserted into an envelope. A label is also automatically printed on the envelope, so civil servants have only to post the envelopes the following working day.

Figure 4. Orchestration of the process of generating a summon



Evaluation

In this section, our experiences with developing a service-oriented architecture are summarized. In Table 2, the evaluation of the quality requirements is discussed. The quality requirements are evaluated using a number of tests. For clarity, it should be remembered that quantitative norms were only available for the performance and availability attributes. From the evaluation of these attributes, the need for service levels control becomes clear. Systems availability is no problem; however, communication and processing time can vary largely. Although a priority mechanism was introduced, it is not suitable for time-sensitive or mission-critical interaction. Response time can exceed 0.5 seconds, when coincidentally a large number of users want to generate documents or if the communication network has a high utilization. The communication manager can provide a solution to this problem as it is constructed using message queuing. The number of document generators can be scaled up, and load balancing can be accomplished by assigning documents to multiple document generators. The testing of this solution was considered to be outside the scope of our research.

A number of experiences with developing a service-oriented architecture are summarized hereafter.

1. *Componentization*: Identifying functionality to transfer to a modular architecture: Document generation is an intuitive appealing functionality that is used by all applications, can easily be reused, and does not need high performance. Presumably, it will be more difficult to find other functionality suitable for modularization.
2. *Asynchronous communication*: When a large number of requests for document generation are submitted a none-response can occur, and the request needs to be resubmitted. Although HTTP is probably so commonly adopted due to its simplicity, the Web services model over HTTP fails as a time-out occurs if it takes more than several seconds to create a response. The resubmission of a request can already be easily handled by the SOAP protocol, however, is not included as a standard mechanism.
3. *Security*: The prototype has no secure communications. sHTTP can be used; however, this protocol is not suitable for creating a complete public key infrastructure. The prototype is used by applications behind a firewall; however, in the future, there is a need for secure communication to create an open market of Web services.
4. *Integrity of transactions*: Functionality to generate a number of single documents and integrate them into one document is not supported by Web services protocols. It is possible to build transaction functionality into the communication manager. There is no standard yet, although Web services transaction (WS-T) might likely become the preferred standard (Dalal, Temel, Little, Potts & Webber, 2003).
5. *Process flow*: The workflow management package, Staffware, is already used to support workflow management. This workflow package can trigger events to start the document generation process. The sequence of Web services invocation by the event-manager was hard-coded in the event-manager. Ideally, a process

Table 2. Evaluation of quality requirements

Quality attributes	Description
User attributes	
Performance	The response time of the document generator can vary depending on the number of requests. At night when an average of 300 summons need to be generated, the response time can increase up to 30 seconds, which would be unacceptable during daytime.
Availability	Availability is no problem. The test indicates an uptime at day- and nighttime of more than 99%. Updating of components had no influence on the uptime; updating of the server can have a minimum impact on the uptime.
Personalization	Generated document can be personalized based on the court and within the court based on the Judge and public prosecutor.
Security	This attribute is neglected in the prototype and needs to be investigated in the future as the Ministry of Justice has a dedicated network. The connection to the Internet is protected using firewalls and they have a dedicated identification and authorization server.
Time-to-market	Law texts are entered as dynamic data in the database. This means that at only one position texts have to be updated to be incorporated in the form. The need for changing workflow to support laws has not been investigated.
Design attributes	
Modifiability	Modular encapsulated generator component. Versions could be updated and changed with ease.
Portability	An XML/SOAP message can be sent from various heterogeneous platforms and applications, including VMS operating systems and Java programming languages to the Windows platform containing IIS server and document generating.
Reusability	The document generator component can and is reused by a number of applications.
Integratability	Communication between legacy systems and document generator is integrated without any major obstacles on the part of the document generation; however, making legacy deploy modular services remains a huge problem as they are not designed for that purpose.
Testability	The component is thoroughly tested, and a number of problems like dealing with multiple requests and making templates fool proof have been resolved.
Use of legacy system	Using the communication manager, legacy systems can be accessed and data can be used by the document generator to generate documents.

orchestration language like BPEL4WS should be used for *both* the workflow management and the document generation process. Staffware and BPEL4WS have differences in order to support routing construct, and the workflows already modeled in Staffware can currently not be converted to BPEL4WS. Van der Aalst (2003) argues that the Web services standardization community did not look at experiences in the workflow domain and that BPEL4WS joins viewpoints from both WSFL and XLANG which makes the language unnecessarily complex.

A number of problems are or can be dealt with by building additional functionality on top of Web services technology; however, this is a cumbersome way. Issues like guaranteed service levels, dealing with delays over HTTP, security, and integrity of transactions seem to be typical functionality which should be handled by standardized Web services technology. In our case study, the middleware layer in the form of the communication manager had to deal with these shortcomings which, however, could lead to limited openness and flexibility in the future. It was decided to delay the adoption of a transaction and process description standards, as they were not viewed as mature and stable enough yet. A major advantage of using the communication manager on the growth path towards a fully service-oriented architecture is that this application can backup and recover data, can give priority to documents needed to be generated, has queues for capacity balancing, can deal with the limited availability of legacy systems, and can provide additional functionality.

Conclusion and Future Research

Rather than replacing legacy systems, modularity can help to leverage investments in legacy systems and incrementally remove functionality from legacy systems. With Web services technology, the life of legacy applications can be extended and a step-by-step migration towards a modular architecture is enabled. Although the current Web services protocol stack can be used for engineering a service-oriented architecture, the current protocols being used in this case study have a number of shortcomings including guaranteeing service levels, security, and data integrity. There are protocols under development to support these problems; however, none of them have become an official standard yet. In our case study, a middleware layer was introduced to overcome these shortcomings and also to add additional functionality.

Currently, integration of business processes and services is an arduous and time-consuming job. The real-time composition of a business process using services is still one step too far. The on-demand buying or renting of services from external parties is even further away. Government organizations can make their applications ready to function in a service-oriented architecture by using standards.

References

Arsanjani, A. (2002). Developing and integrating enterprise components and services. *Communications of the ACM, 45*(10), 31-34.

Armour, F. J. Kaisler, S. H., & Liu, S. Y. (1999). A big-picture look at enterprise architectures. *IEEE IT Professional, 1*(1), 35-42.

Branchau, J. C., Schuster, L., & March, S. T. (1989). Building and implementing an information architecture. *DataBase, 19,* 9-17.

Chen, H. (2002). Digital government: Technologies and practices. *Decision Support Systems, 34*(3), 223-357.

Dalal, S., Temel, S., Little, M., Potts, M., & Webber, J. (2003). Coordinating business transactions on the Web. *IEEE Internet Computing, 7*(1), 30-39.

Fan, M., Stallaert, J., & Whinston, A. B. (2000). The adoption and design methodologies of component-based enterprise systems. *European Journal of Information Systems, 9*(1), 25-35.

Fremantle, P., Weerawarana, S., & Khalaf, R. (2002). Enterprise services: Examining the emerging field of Web services and how it is integrated into existing enterprise infrastructures. *Communications of the ACM, 45*(20), 77-82.

Thatte, S. (Ed.). (2003, May 5). Business Process Execution Language for Web services version 1.1. Retrieved August 19, 2004, from *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/*

Tweney, D. (2002, November). Still waiting for the Web services miracle. *Business 2.0.*

Van der Aalst, W. (2003). Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems, 18*(1), 72-76.

Chapter XVII

Web Services for Groupware

Schahram Dustdar
Vienna University of Technology, Austria

Harald Gall
University of Zurich, Switzerland

Roman Schmidt
Swiss Federal Institute of Technology, Lausanne, Switzerland

Abstract

While some years ago the focus of many Groupware systems has been on the support of Web based information systems to support access with Web browsers, the focus today is shifting towards a programmatic access to software services, regardless of their location and the application used to manipulate those services. Whereas the goal of Web Computing has been to support group work on the Web (browser), Web services support for Groupware has the goal to provide interoperability between many Groupware systems. The contribution of this chapter is threefold: (1) to present a framework consisting of three levels of Web services for Groupware support, (2) to present a novel Web services management and configuration architecture with the aim of integrating various Groupware systems in one overall configurable architecture, and (3) to provide a use case scenario and preliminary proof -of-concept implementation. Our overall goal for this chapter is to provide a sound and flexible architecture for gluing together various Groupware systems using Web services technologies.

Introduction

Since the late 1960s, Groupware aims at supporting various group activities of individuals embedded in multiple teams within organizations as well as between organizations. While some years ago the focus of many Groupware systems has been the support of Web computing, that is, to support access with Web browsers, the focus today is shifting towards a programmatic access to software services, regardless of their location and the application used to manipulate those services. Web services should provide the required standards, protocols, and technologies to fulfil this goal. Whereas the goal of Web Computing has been to support group work on the Web (browser), Web services support for Groupware has the goal to provide *interoperability* between many Groupware systems.

Web services can be seen as a newly emerging distributed computing model for the Web. The standardization process is driven by the growing need to enable business-to-business (B2B) interactions on the Web. Web services are self-contained selfdescribing modular applications. The Web services model develops a componentized view of Web applications and is becoming the emerging platform for distributed computing. The architecture considers a loosely integrated component model, where a Web service interface (component) encapsulating any type of business logic is described in a standardized interface definition language, the Web Services Description Language (WSDL) (W3C-WSDL, 2003). Web service components interact over XML messaging protocols and interoperate with other components using the Simple Object Access Protocol (SOAP) (W3C-SOAP, 2003). Many software vendors and a plethora of standardization consortia, for example, ebXML (EbXML, 2003), W3C (2003), and OASIS (2003), are providing models, languages, and interfaces for the life cycle of Web services: describing, publishing, unpublishing, discovering, and making them available to users for invocation.

Web services coordination middleware needs to support key mechanisms, such as coordination, composition, synchronization, event notification, event logging, transactions, control and data flow, workflow definition and enactment, security, and monitoring management. The basic layers comprising SOAP and WSDL are agreed standards. They provide the means to exchange messages (SOAP) supporting four interaction patterns (Table 2) and to describe service interfaces (WSDL). Higher layers, such as the Web Services Endpoint Language (WSEL), dealing with issues of Quality of Service (QoS) or the Business Process Execution Language for Web Services (BPEL4WS, 2002), the Web Services Flow Language (WSFL, 2003), XLANG (2003), BPML (Business Process Modeling Language) (BPML, 2003), Web Services Choreography Interface (W3C-WSCI, 2003), ebXML BPSS (Business Process Specification Schema) (EbXML, 2003), among others, dealing with Web services workflows, are not standardized yet but only published as specifications. Currently, several BPEL4WS engines are implemented in coordination middleware systems, such as IBM's WebSphere Process Manager or the Collaxa BPEL engine. While XLANG is an XML extension of WSDL describing private workflow processes as Web service composition, WSFL also deals with public models of workflows. XLANG is implemented within the Microsoft BizTalk server. Both of these initiatives of workflow-based Web services coordination do not support existing

normalizations of workflow languages and protocols (for example, workflow interfaces, Wf-XML by WfMC, and CORBA Workflow Facility by Object Management Group). Vertical layers are responsible for foundational services required by all Web services: Discovery (UDDI), Security, for example, WS-Security, Security Assertions Markup Language (SAML, 2003), Transactions, Trust management, routing, Service Level agreements (contracts) to name the most relevant ones, are, however, not yet standardized and currently subject to ongoing research. Current Web services standards mainly focus on horizontal higher layers. Table 1 provides an overview of Web services protocols and their characteristics.

Higher layers of the Web services software stack are built on top of WSDL, which provides four interaction (message) patterns listed in Table 2. The *one-way* pattern provides a high level of decoupling between the requestor and the provider. In the one-way pattern, the request and the response are two messages defined within separate WSDL operations. The request is modeled as an inbound one-way operation, and the response is modeled as an outbound notification operation. Each message is sent as a separate transport-level transmission. The *request/response* pattern also provides a high degree of decoupling between the parties. The service provider needs to be able to handle some application logic; that is, it needs to know the address to which it should send the response. The *solicit response* pattern (also called request/reply with polling) requires four messages defined within two separate WSDL documents to handle an interaction. The initial request is modeled as a request/reply op-eration with two messages, that is, a transmission and one reply sent as a single transport-level exchange. The response is retrieved by a second request. The two operations are implemented as synchronous flows with information being returned from the service provider for each request providing the requester with an acknowledg-ment for each request. In the *notification* pattern (also called request/reply with posting), request and response are handled using four messages defined within two separate WSDL operations. The first request is modeled as a request-/reply operation with two messages sent as a single transport-level exchange. The response is modeled as a solicit/reply operation with two messages also sent as a single transport-level exchange.

Table 1. Web services – characteristics and protocols

Characteristics	**Web service Protocols**
Description	WSDL (Web Services Description Language)
Publishing	UDDI (Universal Description, Discovery, and Integration)
Discovery	UDDI (Universal Description, Discovery, and Integration)
Bind and Invoke	SOAP (Simple Object Access Protocol)
Composition	BPEL4WS, BPML, WSCI
Coordination and Transactions	WS-Coordination, WS-Transaction, WSCI, BPML
Security	WS-Security (proposal by Microsoft), SAML
Routing	proposals (for example, Microsoft)
Service Level Agreements	WSEL (Web Service Endpoint Language)

Table 2. Web services interaction patterns

Message Patterns	Characteristics
One-Way	• Analogous to "Fire-and-Forget" • The message is sent and no response is expected
Request/Response	• Analogous to RPC (Remote Procedure Call) • The sender sends a message and the receiver sends a message (response)
Solicit Response	• Sends a Request without Data for a response • The message is sent and a response is expected
Notification	• Potentially many receivers per message (similar to broadcasts) • Analogous to Publish/Subscribe

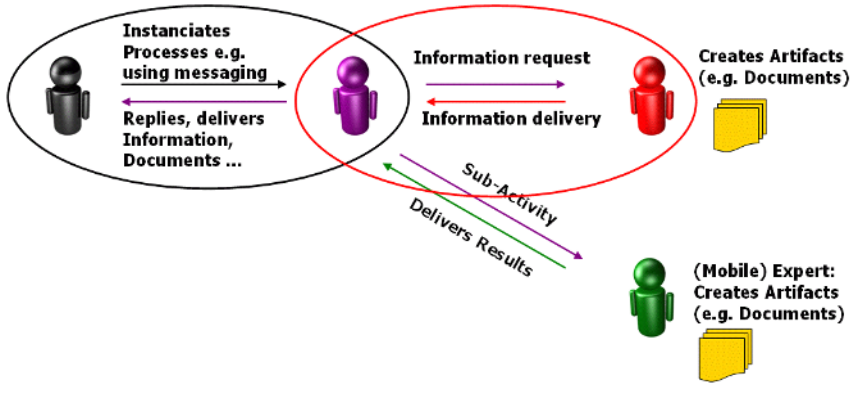
However, not all application servers support all possible interaction patterns so far. The message patterns are the means of interaction between application code using Web services. Hence, the interaction patterns supported by WSDL provide the foundation for coordination services provided by coordination middleware.

Now let us turn our attention to Groupware. Ellis et al. (1991, 1996) provide a functionally-oriented taxonomy of collaborative systems, which assists in understanding the integration issues of workflow and Groupware systems. This classification system provides a framework to understand the characteristics of collaborative systems and their technical implementations. The first category (Keepers) provides those functionalities related to storage and access to shared data (persistency). The metaphor used for systems based on this category is a *shared workspace.* A shared workspace is basically a central repository where all team members put (upload) shared artifacts (in most cases, documents) and share those among the team members. Technical characteristics of Keepers include database features, access control, versioning, and backup/recovery control. Popular systems examples include *BSCW*, IBM/Lotus *TeamRoom,* and the Peer-to-Peer workspace system *Groove* (Groove, 2003). The second category (Communicators) groups all functionality related to explicit communications among team members. Basically, this boils down to messaging systems (e-mail). Its fundamental nature is a point-to-point interaction model, where team members are identified only by their name (e-mail address) and not by other means (for example, by skills, roles, or other constructs as in some advanced workflow systems). The third category (Coordinators) is related to ordering and synchronization of individual activities that make up a whole process. Examples of Coordinator systems include Workflow Management Systems. Finally, the fourth category (Team-Agents) refers to (semi) intelligent software components that perform domain-specific functions and thereby help the group dynamics. An example for this category is a meeting scheduler agent. Most systems in this category are not off-the-shelf standard software. Both evaluation models presented above provide guidance to virtual teams on how to evaluate products based on the frameworks. Current systems for virtual teamwork have their strength in one or two categories of Ellis' framework. Most systems on the market today provide features for Keepers and Communicators support or are solely Coordinator systems (for example, Workflow Management Systems) or are Team-Agents.

Groupware systems have the potential to offer and consume such services on many levels of abstraction. Consider a typical scenario of team work: (Distributed) Team members collaborate by using messaging systems for communications. In most cases, the work-space metaphor is used for collaboration. This means that team members have access to a joint work-space (in most cases, a shared file system), where files (artifacts) and folders may be uploaded and retrieved. In many cases, (mobile) experts are part of such teams and their workspaces. One can argue that a workspace can be seen as a community of team members working on a shared project or toward a common goal. The aim of Groupware systems is to provide tool support for communication, collaboration, and to a limited extent, for coordination of joint activities. Figure 1 illustrates a typical scenario for Groupware usage. It shows that when a person instantiates a process (or project), in most of the cases, e-mail is used since the basis for interoperability for Groupware is limited today to e-mail. However, different group members need to collaborate on (sub) activities for achieving added value and results. Hence, in most cases, other team members create artifacts and send them to other team members who are depending on them to get their own work done. The fundamental problem team members witness is that each person only sees the immediate neighbor (see circles in Figure 1) and therefore lacks complete information about the context of group work and the dependencies and timing constraints. This motivates the need for integration of various Groupware systems with the goals of providing extended context information to all team members and the ability to integrate a great variety of Groupware systems, including mobile systems.

The *contribution* of this chapter is threefold: (1) to present a framework for analyzing three levels of Web services for Groupware support, (2) to present a novel Web services management and configuration architecture with the aim of integrating various Groupware systems in one overall configurable architecture, and (3) to provide a use case scenario and preliminary proof-of-concept implementation example. Our overall goal for this chapter is to provide a sound and flexible architecture for gluing together various Group-ware systems using Web services technologies.

Figure 1. Groupware communities and actors



The remainder of this chapter is organized as follows. The Web Service Management Architecture section presents a novel approach for Web services management architecture based on three levels: business, application, and Teamwork service. Furthermore, it provides an overview on our suggested Web services management and service configuration architecture. The Groupware Support Using Groove section discusses the Groupware support provided by Groove workspaces and shows how the provided Web services interfaces can be utilized for the purpose we present in this chapter. Finally, the last section concludes the chapter.

Web Service Management Architecture

Web Services can be used to address several management aspects as shown in Figure 2:

1. Business management, that is, exposing application functionalities as Web services to other business partners;
2. Application management from a business perspective, that is, wrapping functionalities provided by business applications under a common service interface (Casati & Machiraju, 2003); and
3. Service management, that is, providing common service interfaces to applications for monitoring their operation.

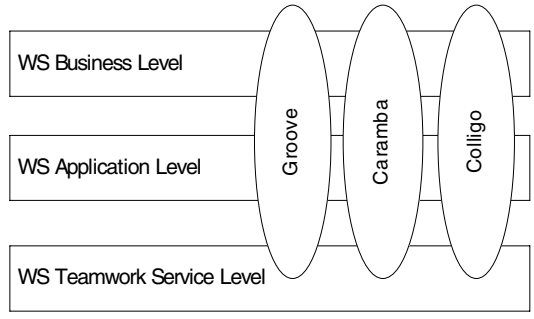
In our case, it is essential that Groupware systems, such as Groove (2003), Caramba (Dustdar, 2004), Colligo (2003), and so forth, have a common way of interoperability. Tools are still closed with respect to integrating their functionality for a particular business. For example, it is not possible to have one team member work with Groove, the other one working with Caramba or some other Groupware tool, and commonly share information and collaborate on a task across specific Groupware systems. The utmost common denominator is to exchange data via some common file formats (for example, XML or XGL) but without any business process support. Hence, virtual teams are restricted to particular tools and tool-specific workspaces or formats.

Web services for Groupware systems for the first time allow such a multitool collaboration in the sense of the above mentioned application management. Uniform Web service interfaces would allow access to Groupware-specific services, such as groups, member data, files, calendars, and so forth, and sharing of these data for a *higher* business value.

Architectural Components

In the description of the key components of our Web Services for Groupware (WS4G) architecture, we focus on the connectivity and process awareness as the basis for WfMS and Groupware systems, consisting of all basic services below the Teamwork Services

Figure 2. Web services levels



layer in Figure 2. Users should be granted access via various types of devices ranging from PCs and notebooks to PDAs or mobile phones for connected, disconnected, or *ad hoc* mode. In the following, we describe the WS4G architecture components depicted in Figure 3.

Participants can be addressed and reached via the concept of a *community* that resembles a project team. This concept allows building communities for specific purposes and tasks as the basis for distributed and mobile collaboration of people. Both participants and artifacts are connected in communities, and users share their information in a loosely coupled peer-to-peer style.

The *User and Community Management* component in Figure 3 provides setup and configuration of community leaders, community members, and also community friends (as a more loosely coupled variant of a team member that could act as a temporary expert or advisor for some task). Adding or removing participants to or from a community, granting participants specific access rights to resources, and so forth, define the responsibilities of this component. It provides a *community* as a central abstraction to other components for addressing groups of people and sharing and exchanging information with them.

Resources cover various kinds of artifacts required for a particular process (or process template) and can be of any MIME-type (text, audio, video, graphics, and so forth). The *Resource Management* component also includes information about particular resources, such as search queries for artifacts, notifications about the availability of an artifact(s) (the peer or server on which it resides), and so forth. In this context, information about a resource includes both meta-information about an artifact and the artifact itself. As a consequence, searches and subscriptions/notifications can be handled on a meta-data level more easily and efficiently for large sets of users.

Process Configuration is concerned with managing the relationships between process participants and artifacts and providing this information to other components. Process participants may be human users or software agents (that is, other software components). Such a process configuration, for example, can be that user (process participant) *Schmidt*

requires the document artifact *paper submission* in a process named *paper review process.*

Process Composition is concerned with managing process models including coordination and synchronization of its subprocesses and tasks. Each process model consists of a set of tasks. The degree of granularity of process tasks can vary. On a generic level, a process model (template) consists of a directed graph consisting of tasks and connection constructors such as OR and AND. On an instance level, a process model consists of instantiated tasks (activities) performed by process participants (human agents or software agents).

Publish/Subscribe, Messaging, and Distributed Search is a component that provides loosely coupled communication among components via messages, events, or synchronous remote method invocations. Its focus is on subscription to all kinds of resources (including artifacts, users, communities, processes, access rights, and so forth). A participant can use this functionality to declare interest in a state of a particular artifact (for example, whenever it is changed or updated, the participant should be notified). The same applies to users, communities, or processes. As a result, this component allows notification of specific activities and can be used for process composition and configuration within or across communities.

Distributed Searches are based on meta-data stored in so-called profiles. These profiles describe artifacts, users, processes, or communities in a concise way and represent it in XML. A distributed search, therefore, queries XML repositories (of different content) on each peer and, if successful, returns the requested piece(s) of information. Distributed Searches further allow querying for information that a user wants to be notified whenever it becomes available; therefore, such queries are stored in the system. Distributed Searches can be further used to search for experts in a particular problem domain and invite them on availability and reachability to join a (virtual) community. This enables the exchange of expertise across communities and processes, which is especially important in mobile and distributed collaboration in large enterprises where people are on the move rather often.

The *Authentication and Access Control* component consists of an access control system called DUMAS (Dynamic User Management System) (Fenkam, 2000) and a security component responsible for integrity, confidentiality, and authentication. The access control system covers three responsibilities: user control, community control, and authorization.

The above Basic Services components are shielded by the MobiTeam Teamwork Services to provide uniform access for teamwork applications. Based on this layer, any specific collaboration application, such as WfMS or Groupware, can configure the Teamwork Services according to their specific requirements and also build new business-specific services on top of the Teamwork Services layer. Such a service configuration, therefore, includes the instantiation of processes (templates) and communities (including artifacts, users, and access rights) for specific tasks (for example, holding a Design Review while process participants are on the move in different branches of the enterprise and/or work on various devices). For more detailed component descriptions, we refer to Dustdar and Gall (2003).

The described component architecture is one major prerequisite to define a set of common services and protocols to exchange all kinds of data among applications. In our case, the applications are different kinds of Groupware systems such as Groove, Caramba, or Colligo. Figure 2 depicts the three different levels of interoperability possibilities among these tools: (1) on the Teamwork Services level based on the above teamwork services component architecture; (2) on the application level for services beyond basic teamwork services, such as calendar, contacts, discussion forum, or tools like shared editing or other means of synchronous communication; and (3) on the business level that is even more specific with respect to the needs of a particular business to use the WS4G architecture to, for instance, run design reviews across different Groupware tools for mobile and distributed process participants all over the world.

Given the common services and interoperability levels, the Web service technology is well-suited to be adopted to work for common services and protocols. For that, we propose the following Web services management and configuration architecture including particular Web service configuration points (Figure 3).

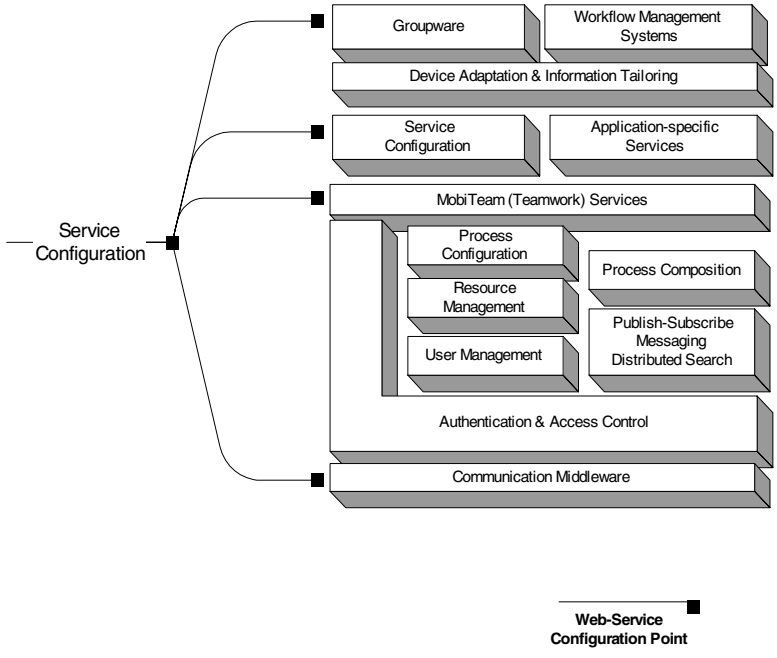
In the architecture depicted in Figure 3, we base on our previous work in which we devised an architecture for distributed and mobile collaboration presented in Dustdar and Gall (2003). There we proposed a teamwork services layer that offers all kinds of *basic services,* such as user and group, artifacts, access rights management, distributed searches, publish-subscribe, messaging, or process management.

The API of these Teamwork Services has been enhanced to a Web service interface to allow usage of such services for all kinds of collaborative systems. For example, basic services, such as group management realized separately in every Groupware system, could then be used in a uniform way. This would exploit the Teamwork Services layer and provide a more general layer as known from communication middleware (for example, CORBA). For Groupware systems, the Teamwork Services will act as a *teamwork middleware* providing the required abstractions and mechanisms for distributed and mobile collaboration scenarios. As depicted in Figure 3, we propose several Web service configuration points:

1. Configuration of applications for business goals (in the sense of business management);
2. Configuration of application-specific services, for example, of Groove (in the sense of application management); and
3. Configuration of teamwork-specific services of a teamwork middleware (in the sense of service manage-ment).

These Web service configuration points allow multilevel customization of Groupware applications. The common denominator of such an approach is the Web service interfaces that enable collaboration across Groupware applications but do not restrict application-specific and, therefore, tool-specific feature sets. As long as each tool maps its features to the teamwork services level, the application (interoperability) level and the business level (for activity-oriented collaboration), the room for unique selling propositions is not limited but, on the contrary, significantly expanded. Interoperability is no

Figure 3. Web services management and configuration architecture



more limited to exchanging data in a common file format but extended to full-fledged intertool collaboration.

In the following section, we describe the Web service support that has been integrated in Groove and, as a first significant step, allows addressing service integration on the application management level.

Groupware Support Using Groove

Groove Workspace (Groove, 2003) is a desktop Groupware software supporting virtual workspaces for working with different groups of people. Groove allows management of projects, file sharing and joint work on files, discussions of work in real-time (for example, audio-conferencing), and presentations sharing. Figure 4 depicts a screen shot of a PDP 2004 workspace, presenting shared files and folders. Each workspace may contain many *tools* to manipulate artifacts of the work-space. Such tools include Files, Calendar, and a Discussion space, in which ideas can be structured and (re)grouped (for example, in

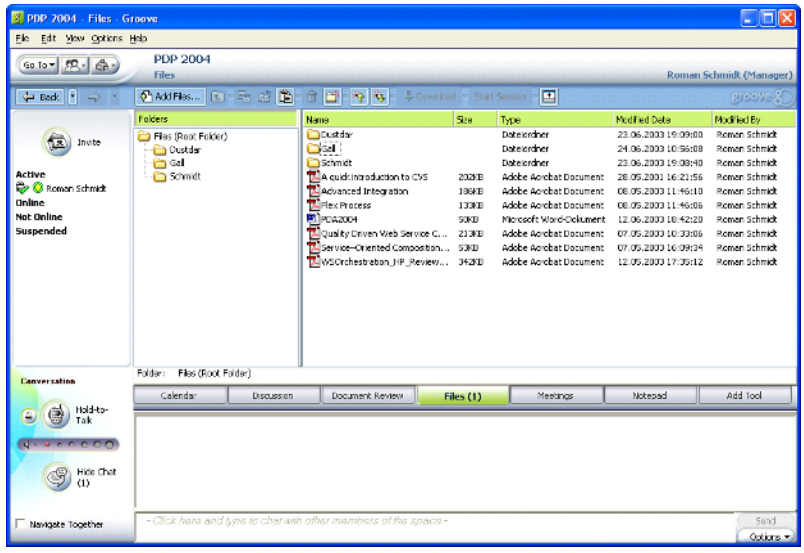
brainstorming sessions). More tools can be integrated by a plug-in concept. However, all plug-ins remain proprietary in the sense that only Groove itself has access to these features.

Groove Web Services

Furthermore, Groove Web services extend the reach of current Groove tools by providing a way to distribute, access, and process Groove workspace data for customized Groupware systems by utilizing standard Web services protocols. Groove Web services allow exposing of Groove objects and data as a Web service, which makes it easier to provide solutions that work in an extended environment. For example, Groove Web services can be used to:

- Integrate Groove tools with external applications running locally on the same device as Groove or running on a server on the network.
- Provide an integrated solution powered by a Groove tool that runs on any endpoint in an IP network, including endpoints running any operating system and lightweight endpoints that include a SOAP client. In addition, Web access to solutions can be provided.
- Allow a Groove user to access their data when they do not have access to a Groove client by using a Web browser or lightweight SOAP client, such as a cell phone.

Figure 4. Groove workspace



- Provide access to data stored in Groove on a Web page on an internal or external Web site.

Groove Web Services Architecture

There are three major components that are part of Groove Web services:

- A SOAP client that consumes Groove Web services;
- A Groove Web services Access Point; and
- A Groove client with Web services enabled.

Figure 5 illustrates these components and the relationship between the SOAP client and the Groove tool.

The remote SOAP client uses the Access Point to connect with Groove clients, the Access Point name registration to identify Groove clients and Groove identities, and polls the Access Point for events fired by Web services. The local SOAP client connects with Groove via *localhost,* accesses the accounts avail-able on the local Groove client, and polls the Groove client via *localhost* for events fired by Web services. The Groove Access Point transmits data between SOAP client and Web services exposed by a Groove client, provides name registration for SOAP client and a Groove client, and provides queuing for the SOAP client. The Groove client contains Groove data and objects, exposes Web services, provides access to data and generates events when data changes, and provides a local access point for SOAP clients running on the same device as the Groove client.

Groove Web Services Development Kit

The Groove Web services Development Kit (GWS GDK) allows developing SOAP clients to access Groove Web services locally or remotely using the Groove Access Point. The GDK includes WSDL definitions of all available Web services, tools, and sample implementations of SOAP clients and documentation. The included tool, Groove Explorer, demonstrates the usage of some Web services. It retrieves information about Identities, Contacts, Shared Spaces, and Tools of a local or remote Groove Workspace, as depicted in Figure 6.

Groove Web services provide a mechanism for SOAP clients to register as listeners for Groove events and check an event queue for messages. The SOAP clients receive events that are generated by changes in the underlying data. These changes can be initiated by the Groove user on the client system providing the Web services, by another Groove user in a shared space, or by the SOAP client itself. Table 3 lists the events provided by Groove Web services provided by Groove 2.5.

Figure 5. Groove Web services architecture

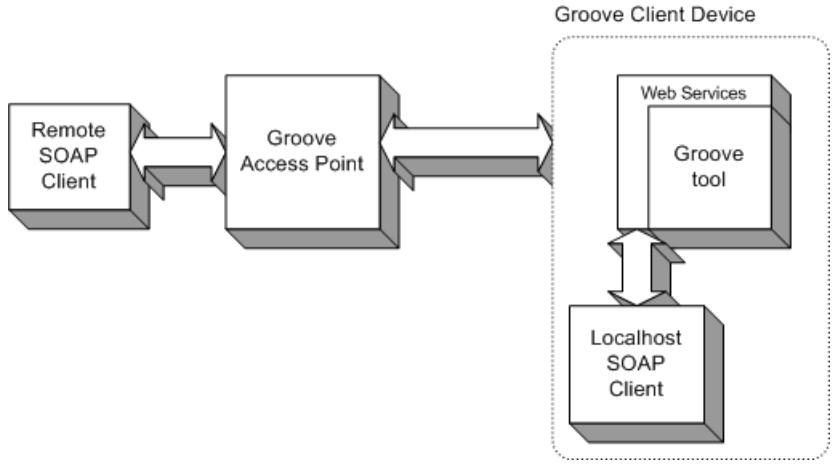
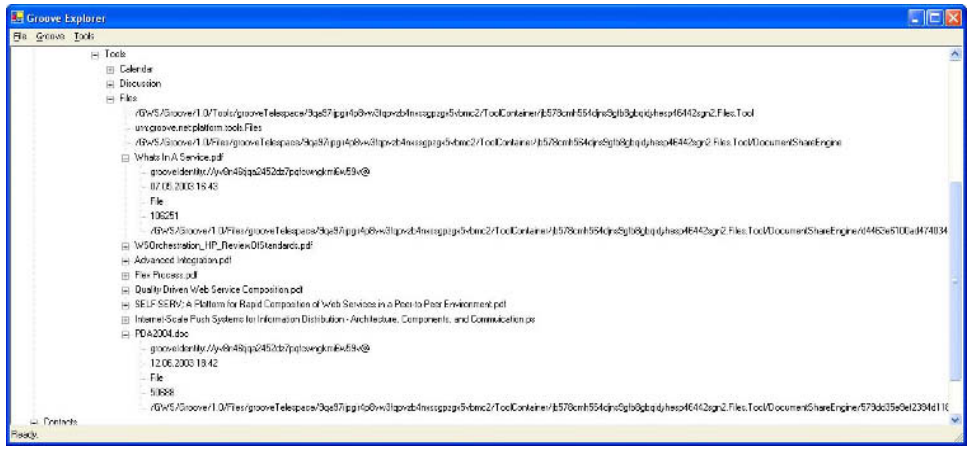


Figure 6. Groove explorer for services



For the Teamwork Services level we adapted the interfaces of our MobiTeam platform, that is, a platform to enable mobile and distributed collaboration among users and communities and should be installed on every peer and, in this respect, has similar characteristics as other Groupware systems: it basically talks only to MobiTeam peers. To overcome this limitation, we raised the level of its interfaces to that of Web services. To showcase that, we show some samples of our MobiTeam interfaces as follows:

```
public interface IMobiteamArtifactManager {
        void insertArtifact(ArtifactProfile artifactProfile, String ID);
        void deleteArtifact(String artifactID);
        }
```

The MobiteamArtifactManager interface allows a physical storage layer access to the artifact repository. Any artifact repository that implements this interface can be deployed ranging from file system for storing the artifacts to an SQL-DBMS by implementing the interface.

Table 3. Groove Web services

Service	Event Class	Event Types
GrooveCalendar	urn:groove-net:CalendarEvent	CalendarAddEventData, CalendarDeleteEventData, CalendarUpdateEventData
GrooveContacts	urn:groove-net:ContactEvent	ContactAddEventData, ContactDeleteEventData, ContactUpdateEventData
GrooveDiscussion	urn:groove-net:DiscussionEvent	DiscussionAddEventData, DiscussionDeleteEventData, DiscussionUpdateEventData
GrooveFilesBase64 GrooveFilesDIME	urn:groove-net:FileEvent	FileAddEventData, FileDeleteEventData, FileRenameEventData, FileUpdateEventData
GrooveSpaces	urn:groove-net:SpaceEvent	SpaceAddEventData, SpaceDeleteEventData, SpaceRenameEventData
GrooveTools	urn:groove-net:ToolEvent	ToolAddEventData, ToolDeleteEventData, ToolRenameEventData

```
public interface IMobiteamCommunityManager {
        CommunityID createCommunity(UserID uid, CommunityID cid);
        void deleteCommunity(UserID uid, CommunityID cid);
        CommunityID getCommunity(CommunityID;
        void linkCommunity(UserID uid, CommunityID subjectId, CommunityID destId);
        void moveCommunity(UserID uid, CommunityID sourceId, CommunityID destId);
        void unlinkCommunity(UserID uid, CommunityID subjectId, CommunityID
sourceId);
        }
```

The MobiteamCommunityManager interface enables all basic services to deal with communities from creating them, structuring them by links, to assigning users to them.

```
public interface MobiteamUserManager {
        void createUser(UserID uid, CommunityID cid, UserID tid;
        void deleteUser(UserID uid, UserID subjectId);
        UserID getUser(UserID actorId, UserID subjectId);
        void linkUser(UserID uid, UserID subjectId, CommunityID destId);
        void moveUser(UserID uid, UserID subjectId, CommunityID destId);
        void unlinkUser(UserID uid, UserID subjectId, CommunityID sourceId);
        }
```

The MobiteamUserManager interface deals with all services concerned with a particular user–creating, deleting, moving, linking, and unlinking to some community.

```
public interface IMobiTeamAuthorisationManager {
            Boolean checkPermission(UserID uid, RightID rid, SubjectID sid, Certificate
 [] c);
            Boolean checkBusinessPermission(string methodPath, Subject[] obj, Certifi
            cate[] c) view);
Certificate [] requestCertificates(XQLQuery query);
            }
```

The MobiteamAuthorisationManager interface covers basic functionality for granting permission for some user to resources; this is only a small part of the interfaces implemented in the MobiTeam platform. Web Services for Groupware that provide the

Teamwork Services functionality can be derived directly from the above interface definitions.

Use Case Paper Review

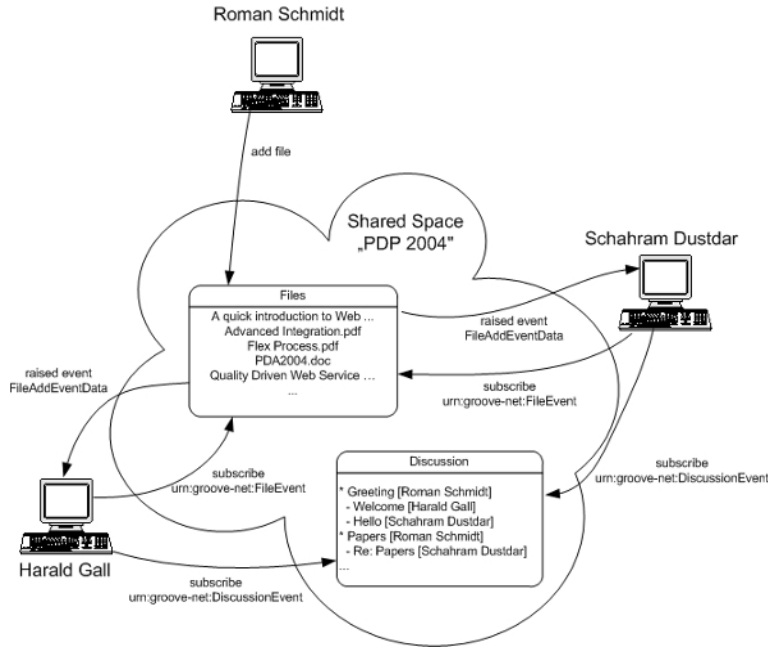
A technical paper review process is used to demonstrate the possibilities of Groove Workspace and its Web service interface. Therefore, a new shared space is created and all reviewers are invited by a review coordinator. The new shared space includes at least the Groove tools, Files to make all papers available for the review process and Discussion to allow a minimum of interaction between the reviewers and the coordinator (Figure 7). Groove Workspace would also provide a special Document Review tool, but as it is not accessible during a Web service interface, it could not be used by third-party software and is restricted to run Groove Workspace for every reviewer.

The used shared space, PDP 2004 in Figure 7, shows the Groove tool, Files, including several papers to review and an empty directory for each reviewer (Dustdar, Gall & Schmidt) which will include the reviewed papers at the end of the review process. As creator of PDP 2004, Roman Schmidt also acts as review coordinator. The reviewers, Dustdar and Gall, are running third party software tools, for example, Colligo for PDAs (Colligo, 2003) or Caramba (Dustdar, 2004), which will interact with Groove by their Web service interface. Therefore, they are not visible to the coordinator by the Groove Workspace. The only possibility for communication between all reviewers and the coordinators is the Groove tool, Discussion, which allows creating topics, writing messages, and writing responses for all participants.

To stay up-to-date, all reviewers using the Web service interface have to subscribe for events possibly raised by the used tools (Figure 7). Dustdar and Gall subscribe the Event Class, urn:groove-net:DiscussionEvent and urn:groove-net:FileEvent, to receive notifications about new or updated files or changes in the discussion forum. For example, if Schmidt adds a new file to the Groove tool, Files, all subscribed listeners receive the raised FileAddEventData event. As Schmidt uses the Groove Workspace software, it is not necessary to subscribe explicitly for events because it is done automatically.

Using subscriptions and events, reviewers will be informed about new papers, or updated versions of papers, and new messages during a discussion. This allows the coordinator to add new papers and assign these papers to reviewers by adding a new topic to the discussion. Therefore, the paper is accessible for the assigned (of course, also for all other participants) and can be reviewed. Afterwards, the reviewed version is added to the Files under the reviewer's subdirectory. Again, all participants will be informed about the progress, and the coordinator can detect the end of the review process.

Figure 7. Groove Web service events



Conclusion and Future Work

Web services are increasingly gaining momentum as an example for service-oriented architectures. Their loosely coupled nature is suited for distributed teamwork where team members are geographically dispersed and utilize various hardware and software infrastructures for their collaborative work. Groupware research has a long tradition; however, today's perception on Groupware, in many cases, only comprises e-mail, whereas the richness of potential Groupware is rarely realized. This chapter presented a novel Web services management and configuration architecture with the aim of integrating various Groupware systems into a coherent and configurable architecture. Furthermore, we provided a motivational example and a small proof-of-concept implementation extending Groove workspaces with Web services. Our future work will focus on extending our previously built Teamwork services platform (Dustdar & Gall, 2003) to provide the presented functionalities for gluing together various Group-ware systems (for example, Dustdar, 2004) using Web services technologies.

References

BPEL4WS. (2002). Business Process Execution Language for Web Services specification. Retrieved August 19, 2004, from *http://www-106.ibm.com/developerworks/library/ws-bpel/*

BPWS4J. (2003). Business Process Execution Language for Web Services Java Run Time. Retrieved August 19, 2004, from *http://www.alphaworks.ibm.com/tech/bpws4j*

BPML. (2002). Business Process Modeling Language. Retrieved August 19, 2004, from *http://www.bpmi.org/bpml-spec.esp*

Casati, F., & Machiraju, V. (2003). *Business visibility with Web services: Making sense of your IT operations and of what they mean to you.* Proceedings of the UMICS 2003 collocated with CAiSE 2003 (pp. 123-135), Velden, Austria.

Colligo. (2003, October 23). Retrieved August 19, 2004, from *http://www.colligo.com*

Conen W., & G. Neumann (Eds.), *Coordination technology for collaborative applications: organizations, processes, and agents* (pp.121-144). Springer-Verlag.

Dustdar, S. (2004). Caramba: A process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and Parallel Databases, 15*(1), 45-66.

Dustdar, S., & Gall, H. (2003). Architectural concerns in distributed and mobile collaborative systems. *Journal of Systems Architecture, 49,* 457-473.

EbXML. (2003, October 23). Retrieved August 19, 2004, from *http://www.ebxml.org*

Ellis, C. A., Gibbs, S. J., & Rein, G. L. (1991). Groupware: Some issues and experiences. *Communications of the ACM, 34*(1), 39-58.

Ellis, C. A. (1998). A framework and mathematical model for collaboration technology. In

Fenkam, P. (2000). *DUMAS: Dynamic User Management System.* Unpublished master's thesis, Technical University of Vienna, Distributed Systems Group.

Groove. (2003, October 23). Retrieved August 19, 2004, from *http://www.groove.net*

OASIS. (2003, October 23). Retrieved August 19, 2004, from *http://www-oasis-open.org*

SAML. (2003). Security Assertions Markup Language. Retrieved August 19, 2004, from *http://www.saml.org/*

W3C-SOAP. (2003, October 23). Simple Object Access Protocol. Retrieved August 19, 2004, from *http://www.w3.org/TR/2001/WD-soap12-part1-20011002*

W3C-WSDL. (2003, October 23). Web Service Description Language. Retrieved August 19, 2004, from *http://www.w3.org/TR/wsdl*

W3C-WSCI. (2003). Web Services Choreography Interface. Retrieved August 19, 2004, from *http://www.w3.org/TR/wsci/*

WSFL. (2002). Web Services Flow Language. Retrieved August 19, 2004, from *http://www-3.ibm.com/software/solutions/-web-services/-pdf/WSFL.pdf*

XLANG. (2002). XLANG specification. Retrieved August 19, 2004, from *http://www.gotdotnet.com/team/-xml_wsspecs/-xlang-c/-default.htm*

Chapter XVIII

Building an Online Security System with Web Services

Richard Yi Ren Wu
University of Alberta, Canada

Mahesh Subramanium
Oregon State University, USA

Abstract

This chapter presents a case study where Web services are used to build a user-centric online security system. It explores complex technical challenges encountered with the use of the Web services and online security technologies. Furthermore, the authors hope that their practical experiences and findings will shed some lights on how the online security system should and can be built in the approach of being user-centric instead of vendor-centric and on the implications of embracing Web services to conventional software engineering processes.

Introduction

Virtually everyone in the IT industry, from vendors to service providers to buyers, has taken up positions to support Web services in their software product and services offerings, but there tends to be some variance in what everyone's definition is. This chapter has chosen the definition from the Web Services Architecture Working Group (2004).

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

In a typical use scenario, one business application sends a request to a Web service using the SOAP protocol over HTTP. Another business service receives the request, processes it, and returns a response using the same SOAP protocol.

For years, the software industry has developed several technologies, such as DCOM and CORBA, to battle against the interoperability problem, but the Internet-scale distributed computing need has pushed those technologies to an absolute limit. Web services promise to meet this challenge. As shown in the above simple scenario, it is simple using simple HTTP-based request/response call patterns using SOAP. It is also loose coupling separating service interfaces from service implementations. It is heterogeneous as the applications and services may be implemented in different languages and operate in different platforms. Finally, it is open because its messaging communication model and service interfaces are based on open standards, SOAP/WSDL.

Though the confidence in Web services has been increasingly gained in the enterprise-computing world, more issues remain. One such issue is whether there is any impact Web services technology has brought about to conventional software system engineering and what it is if the answer is yes. Software system engineering is about tools, methods, processes, and a quality focus (Pressman, 2001); it touches all the issues involved in software system analysis, design, construction, verification, and management of technical (or social) entities.

The case study presented in this chapter is a graduate level development project. Its purpose is to research, design, and prototype an online user-centric security system, called Persona System (Toth & Subramanium, 2003). Conceptually, Persona System is composed of two parts: a Persona Client on the user's device that is integrated with, for example, a web browser, and a Persona Server deployed on a trusted host system. The Persona Server stores and manages the user's personal and identity data. In reality, both parts also need to interact with other online systems, Web services providers and certificates issuing authority systems, for example, http://www.amazon.com.

The implied distributed nature of Persona System introduces a tremendous interoperability challenge simply because both client and server parts are made up of software components that are to be implemented likely in different languages and operational on different platforms using different protocols. This challenge renders itself a perfect case for

employing Web services and exploring online security problems and related system design and implementation issues.

The scale of Web services used in Persona System provides an opportunity for understanding the impact Web services bring about in conventional software engineering realms. Due to our time and resource constraints, we limit our efforts on these realms: architectural design and evaluation, development tools, programming languages, testing, and deployment processes.

The objective of this chapter is twofold: to present practical experience and findings in building Persona System and shed some lights on the implications of employing Web services to conventional software engineering.

Background

Web services do not fundamentally change the conventional software engineering principles as creation of Web services still involves design of services, fabrication of service implementation parts associated with service interfaces, assembly of those parts into a service-based solution, and so forth. However, they do introduce new issues to the conventional software engineering practices and processes. To unearth them, related software development life cycle and methodology need to be well studied with regard to Web services. Due to our limited resources and time, we present our findings only on software architectural evaluation, development tools, programming languages, testing, and deployment processes for the following reasons.

Architecture-First

Architecture-first is the first priority of the modern software management principles argued in Royce (1998) mainly because software architecture is a result of technical, business, and social influences stated in Bass, Clements, and Kazman (1998). Identifying and managing those influences earlier and throughout a software development project is critically needed to build a quality software system. Web services will be a very important part of the technical influences and impacts on how a software project should or can be managed simply because the Web services-based computing effectively and potentially brings in a lot more different project stakeholders involvement, for example, in the case of building Web services-based B2B applications.

Architecture Evaluation

If architecture-first is a first priority principle, how can we determine if one architecture is the *right* one? In our study, unfortunately, very few IT companies have adopted any systematic architectural evaluation methods in order to reach such a delicate but challenging determination. That is possibly because there is none or few out there for them to adopt. In the project, we tried SAAM Method (Kazman, Bass, Abowd & Webb,

1994; Kazman, Bass, Abowd & Clements, 2001) to evaluate the Persona System architecture as we speculate. The dynamic nature of Web services that are not necessarily created with specified functionality as the traditional component-based approach encourages will make such an evaluation even more of a necessity because potentially much broader application of Web services definitely means more technical, business, and social influences.

Development versus Deployment

Our study indicates that the issues related to installing, configuring, running, and integrating Web services do not seem to be encountered until their deployment begins. Plus, the intricacy of customer-side networking environments and operational constraints, testing and deploying Web services are often complicated and error prone without an appropriate mechanism for a good feedback loop from the operations teams to development teams. This, to some degree, seems to challenge modern software development models such as EXP (Extreme Programming) or RUP (Rational Universal Process) and warrants a good understanding of what the challenge may be.

Development Tools

Development tools play a vital role in assuring the success in building Web services-based software systems. We have seen continued changes in the development tools toward a more integrated approach that covers analysis, designs, coding, testing, and deployment. The transition from component-based to Web services-based development will bring in a new engineering issue for both the tools vendors and the developers who use those tools. Tools for Web services-oriented development will likely need to be integrated into an application server or into an operating system. The way such an integration shapes up in the marketplace will somehow impact Web services-based software engineering processes.

Programming Languages

Programming languages carry many resources and technical implications in software development. Web services are clearly redefining the role of programming languages in the system interoperability and, likely, more so with the shift to the virtual machine (VM) approach, such as Microsoft Common Language Runtime. Programming languages, especially compiled languages, are still advantageous to a certain degree as XML-based implementations do not seem to perform as fast as a compiled language code. This is noted in the .NET Framework that provides XML-based Web services and .NET Remoting. Besides, Web services do not seem to practically eliminate all interoperability issues. For example, serializing and deserializing the floating point data type properly between the C# and Java implementation of a particular Web service with the SOAP protocol remains uneasy and more so with user-defined complex data types. Thus, it is important to examine engineering issues related to programming languages.

Context for the Case Study

Building Persona System is a development project jointly participated by three graduate students from Oregon State University and University of Alberta and under the supervision and guidance of Dr. Kal Toth and Dr. Eleni Stroulia. The project started in 2002 and wound down in 2003.

Persona System is based on the concept in Toth and Subramanium (2003) and, unlike Microsoft Passport, provides a user-centric control over the personal data and identity information access using the existing technological infrastructure.

The system has evolved from the basic question: Why should Microsoft or any other company be the custodian of an individual's private data? The discussion was focused on empowerment of users with the proper tools to manage and secure their data. Various systems (including Passport, Liberty alliance, PGP, and TTP) were considered and analyzed for their pros and cons. We were able to draw many advantageous points that the systems have and also pinpoint some glaring defects which make them unsuitable for our purposes. Finally, the requirements for Persona System were identified. One key requirement is to vest the user with proper controls for the management of personal data using public key encryption technologies. Thus, authentication and authorization across multiple disconnected domains with different security policies will need the facilitation of single sign-on and the passing of credentials on the fly. Web services seem to offer a good mechanism to meet this need. The user's credentials and other private data, for example, can be wrapped in XML documents and protected with public key encryption. Credentials in the form of SAML assertions can be issued through Web services.

System Engineering

To meet the project goal we made a few important engineering choices:

Process Model

The process model was informal but the methods and techniques in Rational Unified Process (RUP) and Extreme Programming (XP) were applied whenever possible. For example, the concepts of use cases, architecture-first, and iterations were applied across requirements specifications, design, and prototyping phases.

Architectural Design/Evaluation Methods

Our architectural design approach was based on Kruchten (2001), which presents a 4+1 multiview model, logical, process, physical, and development architectural views cen-

tered around use cases. However, we deviated a little to meet the development needs. First, our logical architectural view is not object model but component model of the design; each component has its corresponding interface, a service interface. Secondly, the deployment architectural view was used instead of physical architectural view. The architectural views were modeled in UML to maintain the semantic and syntactic consistency.

SAAM (Kazman et al., 1994; Kazman et al., 2001) was applied to evaluate the Persona System architectural designs by following the suggested three stages and eight steps. First, a number of nonfunctional requirements was developed from which a set of architectural quality attributes was derived, such as reliability and security. Then, a corresponding set of scenarios against those attributes was determined and, finally, followed by the evaluation.

Development Platforms

All the server-side components were developed on Linux while the client-side components on the Windows 2000 platforms where Microsoft .NET Framework and Compact Framework are available.

Both open source software, such as Apache and commercial software tools like VisualStudio.NET, were used to develop Persona Server and Client parts. WebSphere Studio Enterprise Developer was used for developing a simulated online service provider system for the testing and evaluation purposes.

All the server-side components including the simulated online server provider systems were written either in Java or C/C++. C# was used to develop client-side .NET components. JavaScript was limited only to Web browser rendering.

We did not have an access to any commercial design tool kit such as Rational Rose so completely depended on some basic diagramming tools for communicating our design thoughts and decisions among the team members.

Persona Overview

Persona has been used as a term to denote a user's profile in cyberspace or to denote a software agent in Suziki and Yamamoto (1998). Our usage of the term denotes the entire gamut of data that is in a user's possession. The data can be classified into three types:

1. Personal/Private data: data that belong to the user including name, address, credit card numbers, bank account numbers, and social security numbers.
2. Authentication data: data that allow access to domains protected by authentication schemes (like login names, passwords, and so forth).
3. Authorization data: data attested to by a third party as to the identity/capabilities of the user including a range of certificates and assertions (SAML documents).

Distinction of this data with personal/private data is that the user is the *holder* of the data, and there is an *issuer* who issues it to the user. The assertions may possess specified lifetime and depend on the role the user plays in an organization.

Persona's overriding purpose is to provide direct control over its owner's personal and private data. As a software agent, it encapsulates personal identification, authentication, credentials, and other personal data and exposes selected information to Web services providers on a strictly enforced need-to-know basis. If possible, it should also track where, when, and what data has been left in the care of Web services providers accessed by its owner. This will allow the user to keep data updated at all WSPs of concern. This is to be supported by an agent-based architecture such as Persona with intrusion detection capabilities built in.

Functional and Operational Requirements

The high level requirements identified so far include:

- The Persona Client that is to be embedded in a physical medium like a smart card with a single sign-on (SSO) effect achieved by perfecting an access scheme based on a password.
- The Persona that contains personal data (as decided by the user) and other forms of credentials (support for formats like SAML), X.509 digital certificates and private keys for two-way SSL sessions.
- The SAML assertions and X.509 certificates that are to be issued and signed by Credential Issuing Authority (CA, for short) with a Web services interface to its services.
- Existence of mandatory and discretionary security policies that can be subjected to formal verification methods. The policies are to be formed with respect to domains and the nature of Web services the Persona Client is interacting with.
- Support for credentials from multiple credential issuing authorities due to various affiliations (government, university, employer, and so forth).
- Support for intrusion detection mechanisms.

Conceptual System Model

Figure 1 illustrates the concept of Persona. Users interact with Web services providers and servers hosting Persona Server, which manage their Persona data, and with Credential Issuing authorities. The user may use a variety of devices to get connected. Persona Client manages users' local data and also provides an interface to connect to the services. The Client is often integrated with a Web browser context. Users have a

Figure 1. Persona concept

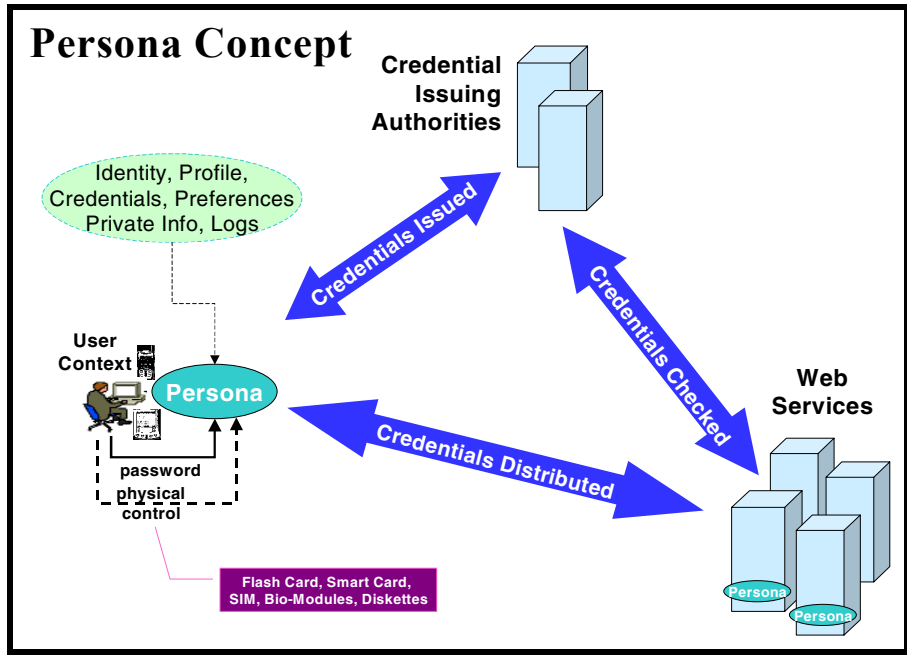


Figure 2. Scenario 1

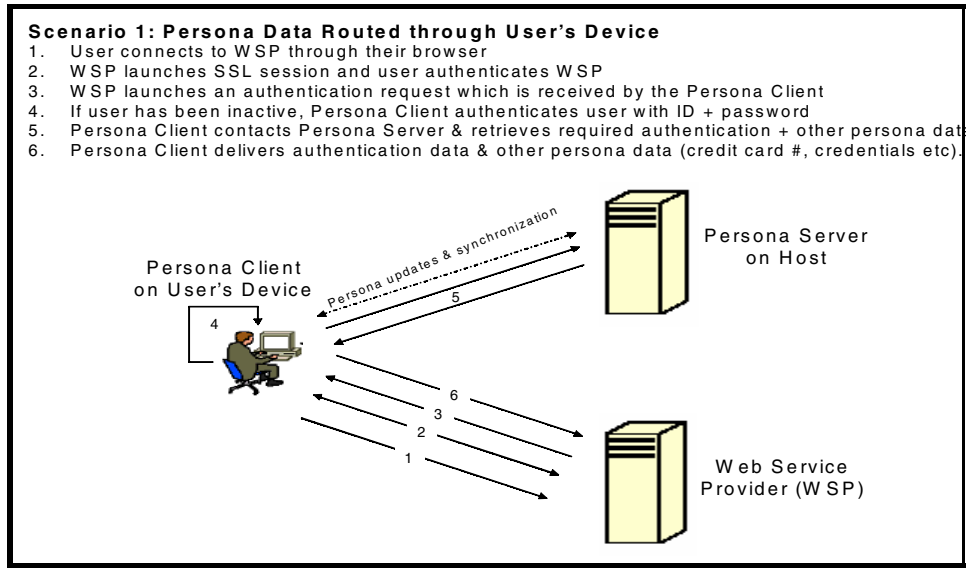
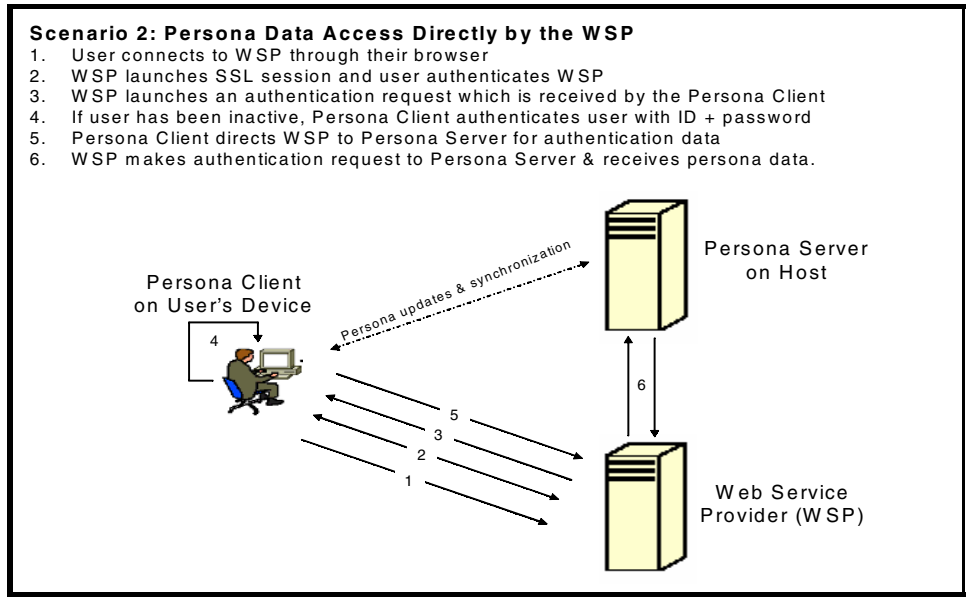


Figure 3. Scenario 2



physical control over the devices. The local data can be protected with a single password. Since the Client is the conduit for connecting to the servers hosting the Persona data and credentials, the use of a single password gives the effect of a virtual Single Sign-on experience. Users use the credentials issued to them by credential issuing authorities as part of their Persona in order to gain access to various domains. The domains that the Persona is associated with have the capability to verify these credentials (depending on their security policy) with the credential issuing authorities. The Web services interface built for this purpose makes this a secure and straightforward process.

Figures 2 and 3 (Toth & Subramanium, 2003) show the two scenarios that illustrate the usage of Persona with Web services. Both of them have a Persona client interacting with a server hosting the Persona data to provide data to Web services on a need-to-know basis.

The data on the Persona is further protected with user-implemented security policies.

Persona System Development

The development mainly involves architectural design and prototype implementation. The architectural design attempts to address a wide range of architectural issues with a focus on their width. The prototyping work is directed to a narrower set of functional partitions of the Persona System. This strategy makes it possible to explore a broad set

of architectural issues and current technologies needed in building Persona System within time and resource constraints.

Design

Architecture Overview

Architecturally, Persona Server and Client form the core of Persona System. Certificates Issuing Authority Server plays a major role in the usage of Persona System but is not part of the core architecture.

Persona Server is entrusted with the storage and protection of user data. It is deployed on a server in which the user possesses some disk space and has access to it. The data stored there is encrypted with a range of encryption schemes available for the user to choose from. The default is a public key encryption scheme. The keys are obtained from the certificate issuing authorities. This approach allows any server to host Persona System, either in full or in part. Our initial development approach is focused on hosting the Persona on a trusted server. Distributed Personae with backup copies for redundancy and with intrusion detection capabilities are something that would be explored in the future.

Persona Server is mainly built with components based on J2EE and Web services. These components are open source-based and interoperable across a wide range of systems. Persona Client is built either with J2ME or Microsoft .NET technology.

The system is designed to be scalable, aiming for a distributed architecture. Based on the amount of data and also considering the need for redundant data, Persona System is to be distributed and stored on multiple servers. Data are to be stored encrypted with public key encryption schemes. We believe that this makes the system inherently secure, even in the eventuality of the hosting server getting compromised. Only users will be able to decrypt and access their own data.

Persona Server Design

Figure 4 illustrates in broad terms the usage of Persona Server. Users obtain signed credentials and other certificates from certification issuing authorities, which are a part of their Persona along with other personal data, and stored in the trusted/semi-trusted host. The interaction with the Persona proceeds with the help of Persona Client that can be integrated with a Web browser context. Both the CA and Persona Server will be hosted using SOAP/Web services to provide open but protected access. Persona Client is responsible for coordinating the interactions with Web services providers, getting the data from Persona Server, and providing them to Web services providers. The back-end storage of Persona data of users can be implemented through commercial database solutions like MySQL, or more preferably (as is done in this project), through more interoperable solutions like XML format documents.

Figure 4. Persona Architecture

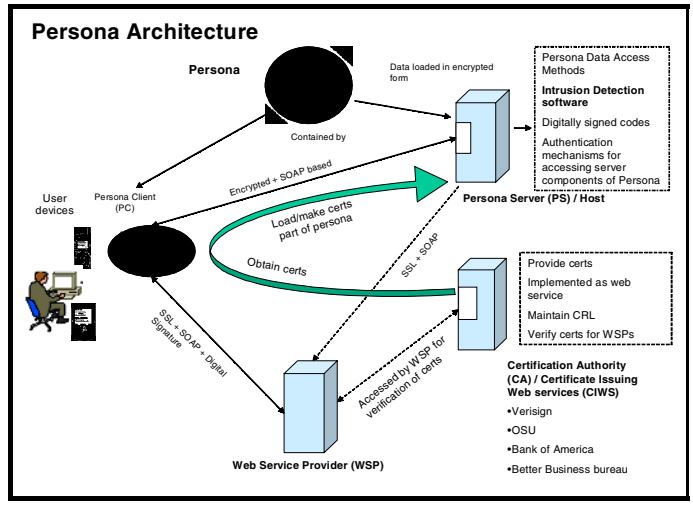


Figure 5. System Architecture

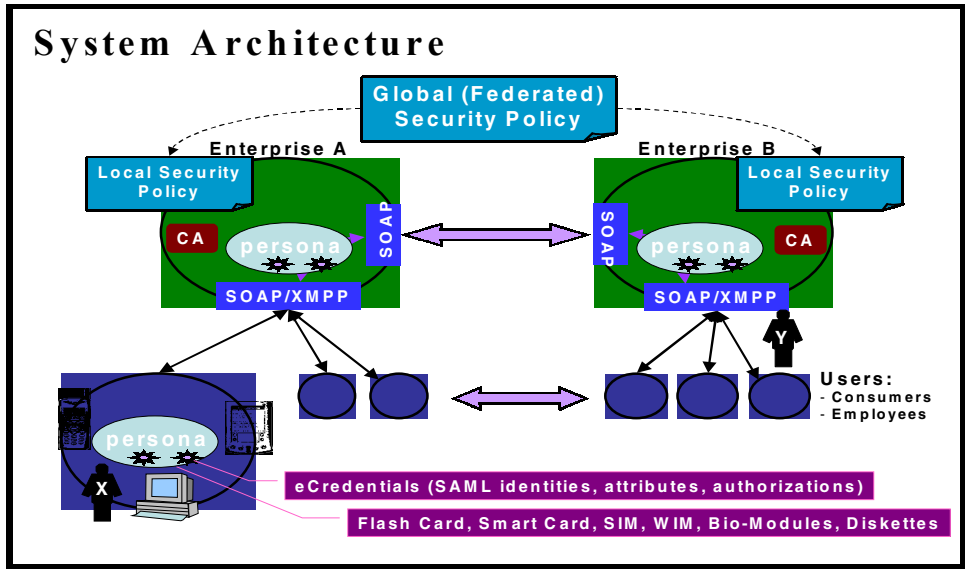


Figure 5 (Toth & Subramanium, 2003) details the usage of Persona across different enterprises and demonstrates the implementation of Web services that enables the seamless interaction among users from one domain to another. The credentials are passed from one domain to another through Web service interfaces.

Major Use Cases

The major functional use cases are:

1. Creation of a new Persona;
2. Add/Delete/Update/Obtain the data in the Persona;

Figure 6

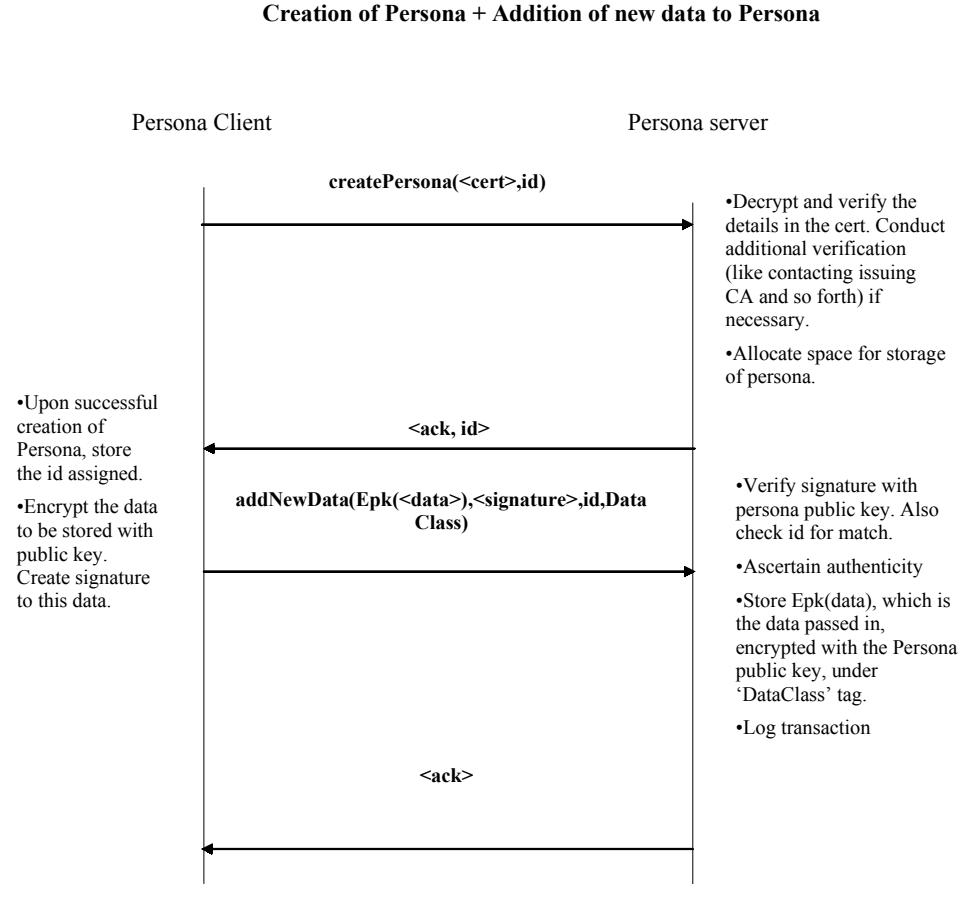
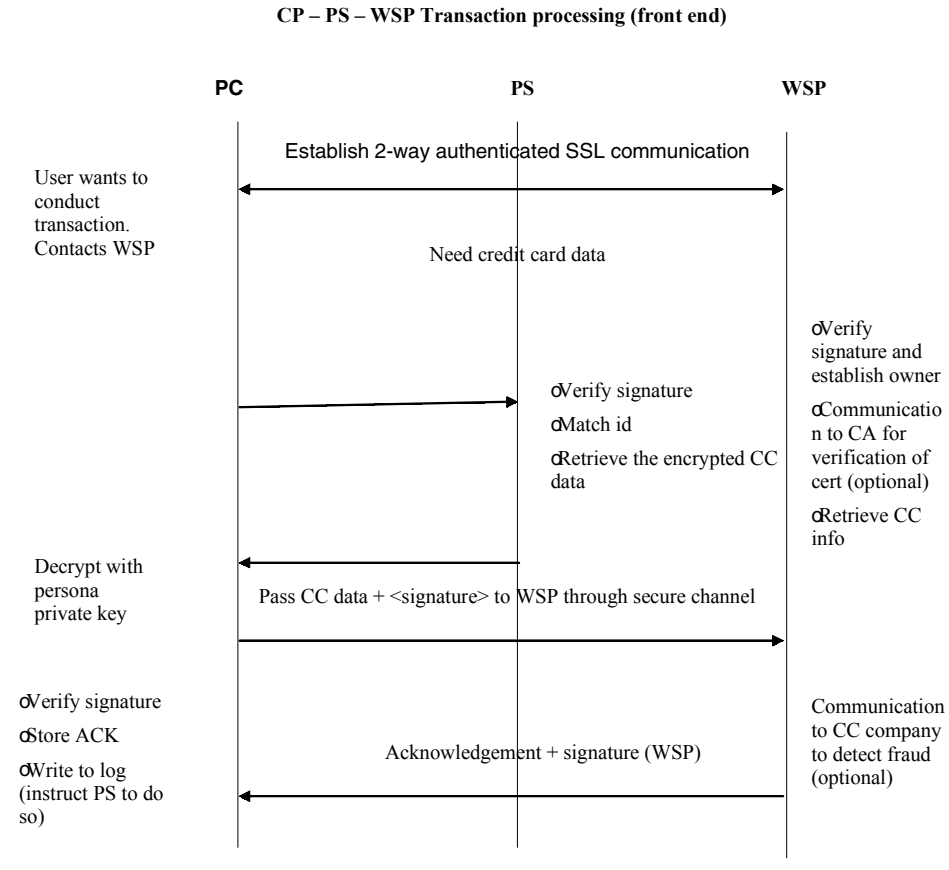


Figure 7



PC –Persona Client

PS – Persona Server

3. Interaction of the Persona with trusted Web services providers without the explicit involvement of the user; and
4. Synchronize the data present in distributed Personae.

Figure 8 and 9 are two sequence diagrams that illustrate how the Persona system works in various scenarios:

The current internal software architecture is derived from the Persona Test Bed configuration shown in Figure 9.

Figure 8. Persona logical view

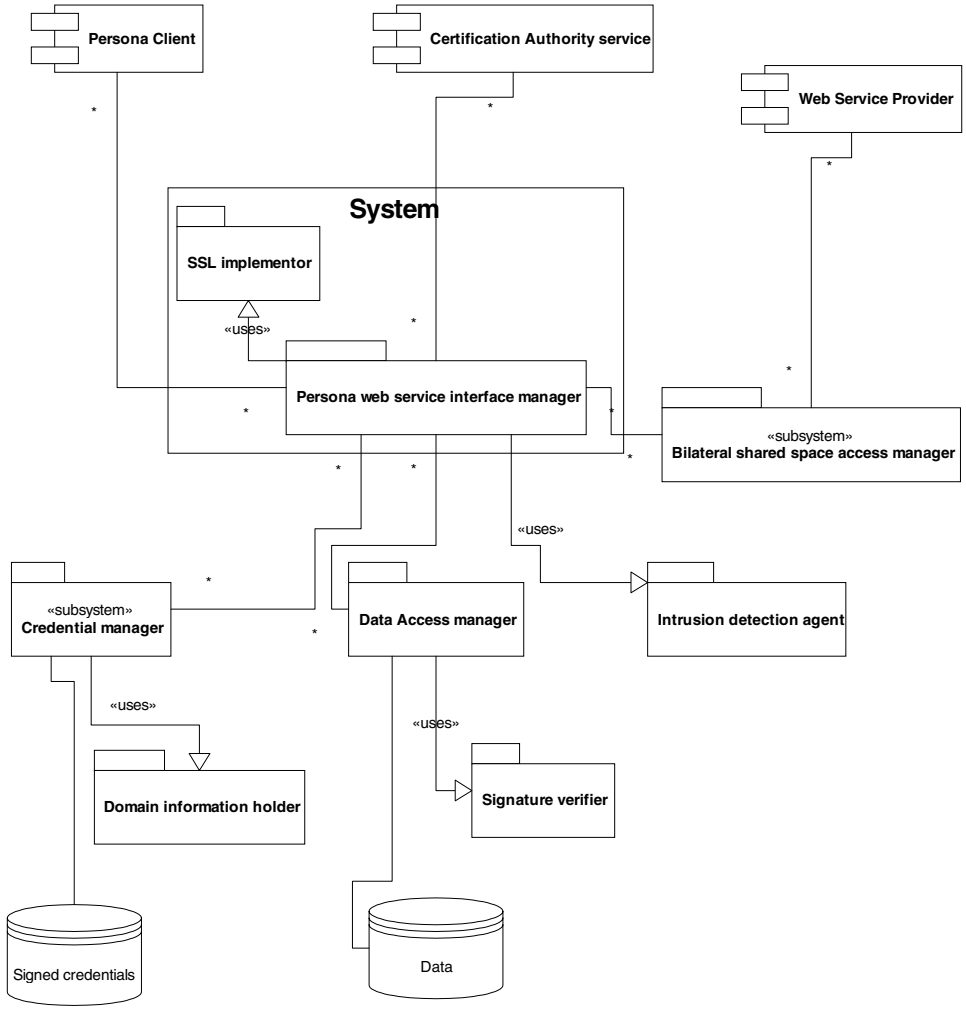
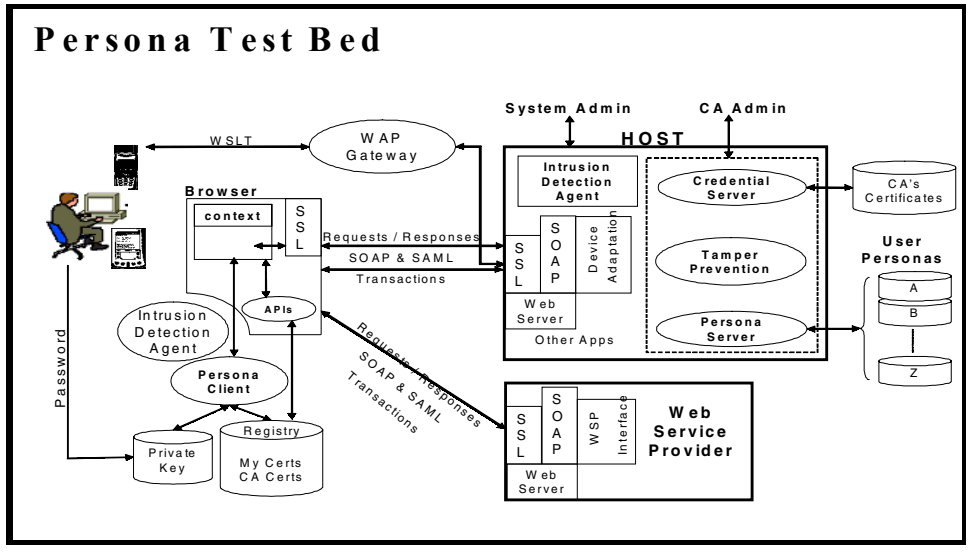


Figure 9. Persona test bed



Persona Client Design

The design involves a Mobile Persona Agent that the end user directly interacts with. It is deployed on users' computing device, such as a PDA, which is connected with their other devices and to Persona Server, certificates issuing servers, and third-party systems over the Internet. The design is mostly driven by major use cases and typical end-user computing environments.

Major Use Cases and Computing Environments

The major use cases are:

1. Access end users' personal and identity data located on a remote Persona Server.
2. Manage end users' own account and data locally on their computing devices operating in a home networking environment.
3. Retrieve or maintain end users' own certificates from a remote Certificates Server.
4. Synchronize the account and data across end users' own computing devices connected with each other.

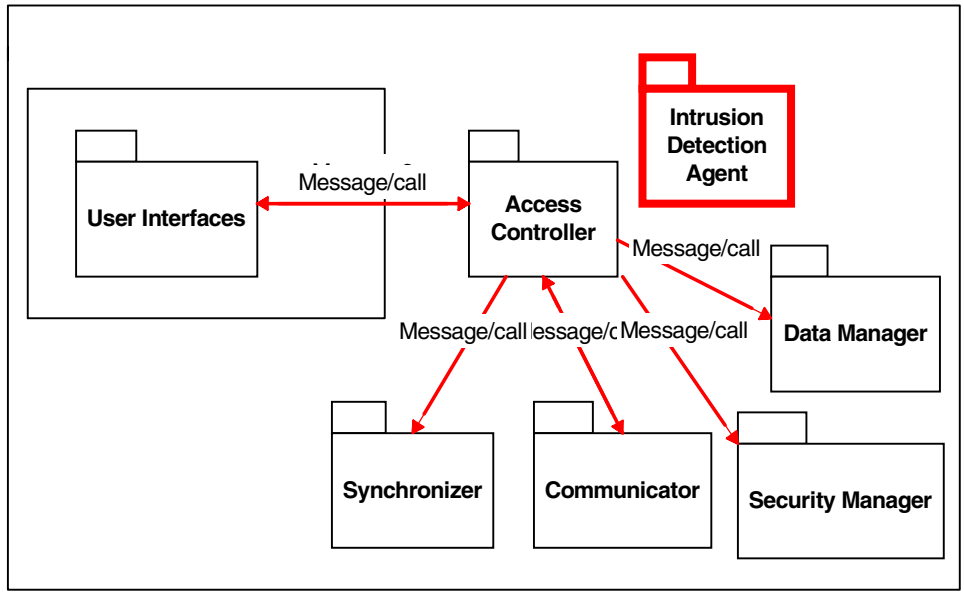
The home networking situation introduces a new challenge for security and privacy of personal and identity data that are either passed through or stored locally on one of those home devices. Given the wireless penetration in the regular user's home and overall networking technology development, we are seeing the growing establishment of a home networking environment where a number of the user's computing (mobile or wireless) devices are connected with each other and to the outside world. Users' own personal and identity data may be stored on a remote Persona Server but may also be managed off-line on their own devices operating in their home networking environments. Mobile Persona Agent (or Persona Client) needs to be designed to address such end users' needs for privacy and security of their personal and identity data in their home networking environments.

Logical View

Figure 10 presents a high-level abstraction of the Mobile Persona Agent.

1. User Interfaces provides the graphical user interface;
2. Access Controller controls calls or events coming from User Interfaces;
3. Data Manager manages the online and off-line user personal and identity data;
4. Security Manager contains the local access security policy;

Figure 10. Mobile persona agent



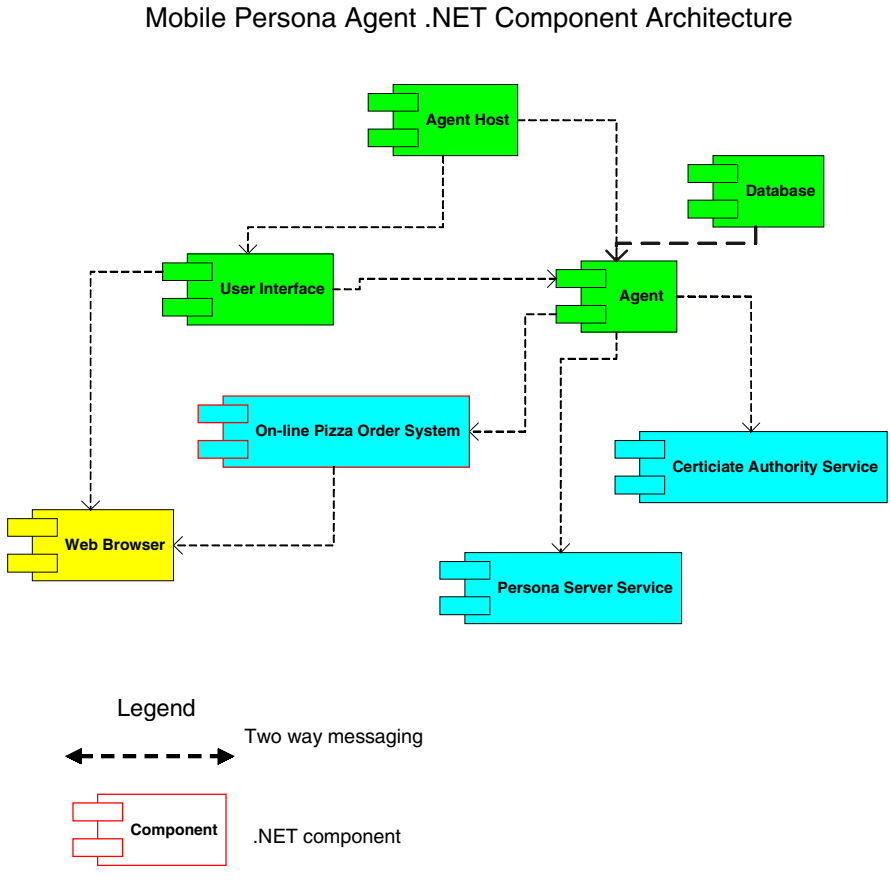
5. Synchronizer manages the Persona data synchronization among all users' computing devices connected with each other in their home networking environments;
6. Communicator contains the Web services logic used to communicate with remote servers or components; and
7. Intrusion Detection Agent detects any unauthorized access to users' devices and Persona data and code and alerts users of any such access.

Component View

Figure 11 presents an implementation-oriented component view.

The Mobile Persona Agent is made up of the following four components:

Figure 11. Mobile persona agent .NET component view



1. Agent Host is embedded in an HTML page that launches the Mobile Persona Agent running in a Web browser that supports .NET Comment Language Runtime but fully under the control of .NET Common Language Runtime;
2. Agent contains all the Persona Client's logic such as Security Manager, Data Manager, and Access Controller mentioned above;
3. Database is an encrypted XML-based document that contains all the user's off-line data; and
4. User Interface provides a graphic way for the user to interact with the Mobile Persona Client.

Prototyping Implementation

We have not implemented the Persona System fully according to the architectures we have developed. Instead, we have prototyped it with the focus on a few areas to further our understanding of the Persona System requirements and the degree to which the current technologies can support the Persona System needs. To facilitate this prototyping need and test the Persona components, two external systems are designed and prototyped:

1. Certificates Issuing Server
2. Online Pizza Order System

Web services are the only way for communication between Mobile Persona Agent, Persona Server, Certificates Issuing Server and Online Pizza Order System with SSL support.

Persona Server

The server was developed on Red-Hat Linux. Apache Tomcat Axis was used for hosting the Web services for the Persona server. The Web services were initially developed using Java Web services. Axis automatically locates the Java Web service code files, compiles the class, and converts SOAP calls correctly into Java invocations of the service class. Java2wsdl can be used to generate WSDL files from Java programs.

A client test code was developed just to test those Web services in isolation before the Mobile Persona Client code was ready for integration and testing.

Certificates Issuing Authority Server

The server is responsible for issuing signed certificates and credentials to requesting users. It employs Web services for exposing such services as requesting a new certificate, validation, recovery of lost certificates, and so forth. Those services are associated with security policies depending on how secure the certificates need to be.

The server was implemented on a Red-Hat Linux system both with a Web page interface, where a form is presented to the user, and with a Web service interface, implemented with Java Web services. The Web page interface uses Apache 2.0 with mod_ssl and openssl 0.9.6b libraries. The page was implemented with Perl scripts.

Online Server Pizza Ordering System

To test the interoperability and interactions between the Persona System parts and online Web services providers, a simulator was built with IBM WebSphere Studio Enterprise Developer 5.0 and deployed and configured on IBM WebSphere Application Server running on Windows 2000 Server. The simulator provides a number of Web services in WSDL that allows Persona Server, Certificates Issuing Server, and Mobile Persona Agent to communicate directly with each other for a particular online transaction, for example, ordering pizza.

Users place a pizza order over the Internet. Before they commit to the order, they are asked to submit the payment and payment method, which triggers a series of actions, involving authentication, authorization and payment determination, and transaction. These actions via Mobile Persona Agent all occur on Persona server systems and Online Pizza Order System with Web services and SSL.

Mobile Persona Agent

The client was prototyped using Microsoft .NET Framework. The code is literally a set of .NET C# assemblies that are downloadable from a remote Web site. It is then loaded in Internet Explorer but effectively executed under the control of .NET Common Language Runtime.

The Agent performs those functions described in the previous section and communicates with Persona Server, Certificates Issuing System, and external online Web services providers via Web services calls over SSL.

Work Evaluation

The following sections present the evaluation on the Persona System itself and system engineering.

Persona System

The evaluation of the Persona system focuses on the design, testing, and deployment of server and client components and the interactions with the certification authority system and Web services.

Architectural Design

The evaluation of server and client components are entirely under different criteria. Server components are evaluated based on their robustness and scalability and also their resilience under different load conditions. Threat analysis and counter measures are another gamut of evaluation conditions for the server components. Client components are mostly evaluated under the heads of ease of use, robustness, error handling, and effective communications. The design is aimed to be as loosely coupled as possible. We have found the use of UML an effective way to bring out the design decisions and depict the interactions among components (server, client, certification authority), both intra- and inter-, and also helps to show the process flow in the different design elements and in different processing phases.

Deployment

Deployment of actual systems is to be evaluated under ease of use and also how quickly changes can be made. We have found that with Web services, deployment is a very straight-forward process. Changes are reflected quickly. Of course, as we go to more complex systems, dependencies increase and so do the amount of associated change. This is truer of the server components. The server has components to interact with clients and also with Web services which request data. The server has to build different levels of security depending on the data it stores, and consequently, this leaves the door open to different levels of authentication. The most basic kind of data can be provided after the verification of a simple signature, but more sensitive data requires additional authentication like in the form of SAML signed assertions/X.509 certificates.

Client components have much more flexibility with regard to deployment. The components can be built as simple applets or small applications in the user device. Also, since all of the development is done with open source tools, deployment and execution of programs is very straightforward.

System Engineering

The system engineering work evaluation is focused on architecture evaluation methods, the usage of open source code, programming languages and development tools, and testing and deployment.

Architecture Evaluation Methods

The architecture-first strategy proves to be even more important and necessary for developing Web services-oriented applications. Service orientation makes it possible to partition the Persona System along the Web service layers or interfaces earlier in the project cycle. The partitioning has allowed each team member to start their development

independently and earlier with a clear focus more on the service boundary or interface issues rather than the implementation of each other's partitions and resulted in easier coordination among each other's design, coding, and testing work.

Multidimensional architectural views, such as component and deployment views, are critical with the use of Web services as Web services deployment depends much on the customer's internal networking systems and operations, in our case, on the .NET/ Windows 2000 and Apache Tomcat/Linux networks in the intranet and over the Internet. Addressing deployment architectural issues earlier in the game has made it possible to address more effectively phased testing and operational issues earlier, as well.

Architectural evaluation, though still difficult, is absolutely necessary with the Web services-based software engineering. SAAM applied in our project seems to work reasonably well to address those nonfunctional quality attributes. However, the evaluation is still manually driven, very time-consuming, and definitely needs some auto-tool to be productive.

Open Source

The open source code and tools from http://www.apache.org were widely used in the project. They are readily available at little cost. For research and development like this project, open source code and tools are affordable and indispensable. However, compared with the commercial tools like IBM WebSphere Studio Enterprise Developer and Microsoft Visual Studio.NET, the open source code and tools are not productive enough because of lack of *high app,* modeling, and code generation capabilities.

Development Tools

Easy-to-use development tools is one of the keys to build quality Web services-based software applications or products. Both Microsoft Visual Studio.NET and WebSphere Studio Enterprise Developer support the Web services-based software development and provide fundamental services and facilities for Web services test and deployment and version control. Both tools have some type of wizard to somehow automate the process in generating Web services and related code, assembly, testing, and deployment.

One issue with the commercial software development tools is they are not only platform-dependent but also vendor-centric when dealing with further integration into mainframe, database management products, packaged application suites, application server products, and pure middleware systems. Microsoft Visual Studio.NET provides superior services and capabilities for .NET component integration with DNA/COM components that only run on Windows platforms. IBM WebSphere Studio Enterprise Developer provides similar superior hooks and adapters into IBM-only systems.

Programming Languages

With Web services in place, programming languages do not play a significant role in addressing interoperability issues. We do not need to worry too much about the constraints imposed by a particular programming language, C# or Java. However, the interoperability issue remains with user-defined objects or components and objects, such as Date and Floating Numbers. The other issues with programming languages are related to supporting Web services stacks. For example, there is no easy way for a C# assembly to access the UDDI services implemented in Java and vice versa, and it is still cumbersome to resolve the difference between the WSDLs generated in VisualStudio.NET and in the J2EE-compliant tools because the former bundles the Web service interface together the Web service implementation but the latter does not.

Deployment

The two major engineering issues we have observed is that Web services do not effectively begin to encounter interoperability with third-party systems until after their deployment starts and that developers are not necessarily clear about how those Web services will be eventually called. The first challenge is more related to the fact that most of the issues related to Web service installation, configuration, and runtime are not encountered until after deployment. The second challenge is because as soon as Web services are published, it will not be easy to control who should access them; quite often, the types of access may not be well defined in the development.

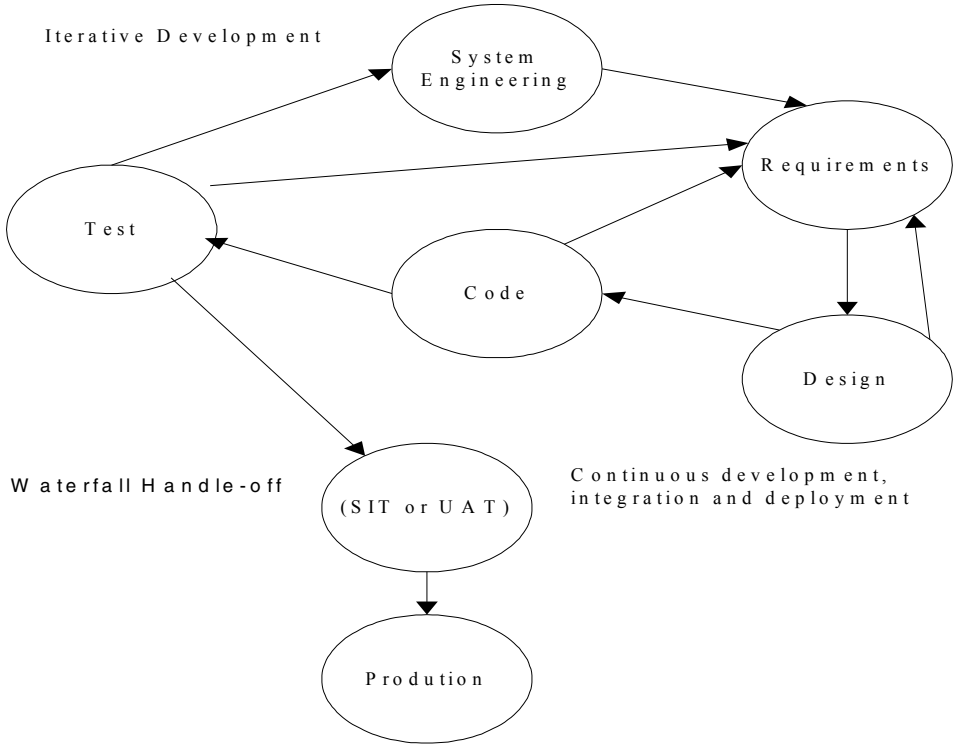
To address those problems, developers and operations teams will need to work much closer, likely enabling developers to debug interoperability problems and test fixes even on the live systems but in a secure and controlled manner

Process Model

XP and RUP that we attempted to apply are good but not complete in addressing some of the engineering issues, in particular, when it comes to addressing the needs of Web services. For example, RUP assumes building code from a domain model derived from use cases, which is incomplete for Web services. XP does not really define a clear way to handle the new testing and deployment issues. For example, how to manage integration on customers and handoff to a production environment/team to operate.

Figure 12 presents our experience in the Web services development. The model is iterative in development but Waterfall after the handoff to production.

Figure 12. Process model for Web services



Future Trends

The Web service technology has advanced rapidly with a continuous push from the major business drivers, such as increased market share, quick and easy business process integration and adaptability, and centralized, real-time availability of services regardless of platforms. In the following sections, we outline a few important changes on the horizon and engineering challenges ahead.

Changes on the Horizon

Personal/Private Data Management and Accounting

As we have seen, personal data management and accounting for where data have been left and in whose care are all gaining prominence among Web-literate consumers.

Moreover, consumers are getting more interested in how Web sites use their personal data and how it is protected. With the evolution of complex authentication systems to enable user migration across diverse systems and environments, secure storage of profile information, minimizing the number of user log-ins to different services to enhance user experience, and storage and transfer of user preferences across diverse systems and devices are focal points for companies intending to provide better service. The explosion of data is causing a security risk that needs efficient handling. However, better security solutions can be created with open source and free resources like SAML, XACML, and so forth. Though much is left desired in the forms of universal standards in Web services, the healthy stage of development of many open source security tools is a positive outcome.

Evolving Web Services Protocol Stack

Since Web Services Conceptual Architecture 1.0 in Kreger (2001) was published in May 2001, the stack of Web services protocol has continued to evolve. Lawrence Wilkes presents a more updated picture of the current Web services protocol stack (http://www-106.ibm.com/developerworks/webservices/library/ws-ref2/#f1).

With the rapid evolvement of Web services protocols, the degree of industry consensus on low-level Web service protocols has been significant. For example, there is a general consensus on low-level protocols such as SOAP and WSDL. However, the road does not seem to be so easy and smooth in the agreement on the high-level protocols. For example, there are currently two alternative proposals, namely, Reliable Messaging and Orchestration. The alternatives reflect an IBM/Microsoft-led initiative on one side and one led by Sun/Oracle on the other.

Though alternative proposals have been made in some high-level protocols, it seems that the formation of an appropriate working group in either W3C or OASIS has usually seen the attempt for convergence or subsequent convergence of all interested parties.

The proposal of various Web services protocols has been a fast moving area but their transition into actual well-accepted open standards turns out to be unbelievably slower. Currently, the areas not fully addressed are Web services management, service/business level agreements, and WS-Security.

Open Source Movement

Open source code and tools continue to grow and will be an important part of the commercial development tool offerings over time. There are clearly signs that the Apache KDE and Gnome are rapidly developing Web service hooks and tools for serious industrial uses. Tomcat and related Web services library are included in the IBM WebSphere Web Service Development offering.

Model-Driven Architecture

Model-Driven Architecture (MDA), driven by the Object Management Group (http://www.omg.org/mda/) describes how to create standards-based, technology-independent models of business concepts and then map them to different specific technologies. Platform-independent applications built using MDA standards can be realized on a range of open and proprietary platforms including CORBA, J2EE, .NET, and Web services. The key concepts in an MDA model are business information model and service model. The MDA approach may potentially alleviate some issues we encountered in the architectural evaluation.

Better Development Tools

Development tools continue to play a more important role in the success of Web services-based software development as better tools can extend Web services to provide functionality and ease of use that will map business processes to application functions in a seamless fashion, regardless of platforms, languages, or devices. Integrated development environments (IDE), such as Microsoft's Visual Studio .NET already provide a core framework to leverage, create, and consume Web services in an integrated fashion. Other integrated development environments such as IBM's WebSphere are extending their J2EE environments to incorporate Web services into an easy-to-use development platform. In the not-too-distant future, it will be possible to discover, bind, and execute Web services dynamically and to coordinate the process flow through one integrated development environment. This will allow external Web services to be widely adopted, which is very important as the need for service level agreements (SLAs), confidentiality/security, and alternative suppliers is a must for any business providing or consuming Web services.

Programming Languages

The battle for the enterprise market will mostly remain between Java and .NET, both of which rely on a virtual machine (VM), the layer of abstraction between the programmer and the operating system. Much of the power of Java and .NET is not in the programming languages themselves but in services made possible by the VM. These services include runtime-type information, security, versioning, mobile code, and dynamic code generation.

Engineering Challenges Ahead

The continuously evolving Web services protocols introduce an amount of uncertainty as software development cannot wait for the completeness of a particular protocol. For

example, the WS-Reliable Messaging is not ready and incomplete but the Web services-based software system to build still requires messaging reliability. A first work-around may be with the use of MQSeries, CICS, or MSMQ/COM+, when it is not ready and then incorporate it later when ready.

Coarse-grained Web services will pose a big challenge to many already existing objects and components based on J2EE, CORBA, or COM because they are generally fine-grained. It is clearly the business interest to reuse and protect those technology-specific objects and components, but it will likely take years to complete this First Generation Web Services Integration journey as described in Frankel (2003).

Conclusion

It is clearly necessary and important to empower users with the proper tools to manage and secure their personal and identity data over the Internet. Web services, public key encryption, and related XML technologies, such as SAML, offer much promise to address this need, but providing a well-working solution based on those technologies has proved to be a tremendous challenge. Our work in building a user-centric online security system is far from over, but our findings and practical experience have enabled us to get closer to delivering a system that will finally find a user marketplace.

Web services do not fundamentally change the software engineering principles but introduce some issues that need to be well understood in order to build high-quality systems. To unearth those issues, our case study has been focused on architectural design/evaluation, development tools, programming languages, and overall processing model.

We have presented our findings in the Persona System development project. Architecture-first strategy proves to be very important and necessary for architecting Web services-based software systems. Development tools are increasingly critical in the success of Web-services-based software development. Open source continues to play a key role in the Web services-based software development. Programming Languages remain important with the interoperability issue. The conventional software testing and deployment model and practices are challenged as development is more deployment-centric.

Web services, derived from service-oriented architecture, will continue to evolve rapidly in many aspects. Newer issues and challenges will emerge and further challenge the conventional software engineering methodology and practices. Thus, a further study will be needed not only to understand those newer software engineering issues but also to discover best practices and formulate appropriate development process model.

References

Bass, L., Clements, P., & Kazman, R. (1998). *Software architecture in practice.* Addison-Wesley Longman.

Frankel, D. S. (2003). *Applying MDA to enterprise computing.* Wiley.

Kazman, R., Bass, L., Abowd, G., & Webb, M. (1994). *SAAM: A method for analyzing the properties software architectures.* Proceedings of the 16th International Conference on Software Engineering.

Kazman, R., Bass, L., Abowd, G., & Clements, P. (2001). *An architectural analysis case study: Internet information systems.* University of Waterlool, Carnegie Mellon University and Georgia Institute of Technology.

Kreger, H. (2001). Web services conceptual architecture 1.0. IBM Software Group. Retrieved August 19, 2004, from *http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf*

Kruchten, P. (2001). Architecture blueprints: The "4+1" view model of software architecture. Retrieved August 19, 2004, from *http://www.rational.com/media/whitepapers/Pbk4p1.pdf*

Pressman, R. S. (2001). *Software engineering: A practitioner's approach* (5th ed.). McGraw-Hill.

Royce, W. (1998). *Software project management: A unified framework.* Addison-Wesley Longman.

Suziki, J., & Yamamoto, Y. (1998). *Document brokering with agents: Persona approach.* Proceedings of the JSSST WISS'98 Workshop on Interactive Systems and Software.

Toth, K., & Subramanium, M. (2003). *The Persona concept: A consumer-centred identity model.* Proceedings of the MobEA Conference, Budapest, Hungary.

Toth, K., & Subramanium, M. (2003). *Requirements for the Persona concept.* Proceedings of the International Workshop on Requirements for High Assurance Systems, Monterey, California.

About the Editors

Zoran Stojanovic is a researcher at the Faculty of Technology, Policy and Management, Delft University of Technology, The Netherlands. His research interests are in the areas of component-based development, Web services, system modeling and architecture, geographic information systems (GIS), and location-based services. He received his graduate engineering degree and master's of philosophy degree in computer science and GIS from the Faculty of Electronic Engineering, University of Nis (Yugoslavia) (1993 and 1998, respectively). He has been working since 1993 as a researcher and lecturer in computer science, software and system engineering, first with the University of Nis, Yugoslavia and after February 2000, with the Delft University of Technology. During this period, he has been an author of a number of publications.

Ajantha Dahanayake is an associate professor at the Faculty of Technology, Policy and Management, Delft University of Technology, The Netherlands. She previously served as an associate professor in the Department of Information Systems and Algorithms at the Faculty of Information Technology and Systems. She received a BSc and MSc in computer science from the University of Leiden and a PhD in information systems from Delft University of Technology. She had served in a number of Dutch research and academic institutions. Her research interests are distributed Web-enabled systems, CASE, methodology engineering, component-based development and m-business. She is the research director of the research program, Building Blocks for Telematics Applications Development and Evaluation (BETADE).

About the Authors

Hemant Adarkar is currently chief technology officer at Ness Global Services, India. Dr. Adarkar holds a master's degree in electronics and a PhD in experimental nuclear physics. He has more than 18 years of experience in heterogeneous systems. In the last 10 years, Hemant has been involved in architecting secure, high transaction volume systems primarily in the BFSI industry. His areas of interest include information security, application integration, mobile computing, and service-oriented architecture. He has authored several papers in international journals and conference proceedings and delivered talks around the globe. He is an examiner and a guest faculty at various technical and business schools of international repute.

Mikhail Auguston is an associate professor with the Computer Science Department of the Naval Postgraduate School in Monterey, California, USA. His research interests are in testing and debugging automation, programming language design and implementation, and visual programming. Recent work includes debugging automation tools based on event grammars and computations over event traces, the compiler writing language, RIGAL, and experimental visual programming notation.

Boualem Benatallah is senior lecturer at the University of New South Wales, Sydney, Australia. His research interests lie in the areas of Web services, workflows, Web semantics, and mobile data management. He has several ARC (Australian Research Council) funded projects in these areas. He was a visiting scholar at Purdue University, USA and Visiting Professor at INRIA-Loria, France. He is member of the editorial board of the *International Journal of Business Process Integration and Management.* He has been a program committee member of several conferences and has published widely in international journals and conferences.

Keith H. Bessette holds a bachelor's degree and master's degrees in computer science and engineering at the University of Connecticut, USA. His research interests are network security and mobile data mining. Mr. Bessette works as a network security engineer for UITS, University Information Technology Services, at the University of

Connecticut, responsible for network intrusion detection, intrusion prevention, and vulnerability analysis and testing.

Benoit Bloin was formerly a research assistant in the Computing Department at Lancaster University. His research collaborations included Extended Component Architecture Design and Management (ECOADM), a European Union sponsored project whose aim was to develop tools and methods to support the development of component-based systems. He now works for Hyperion Solutions, UK.

Barrett R. Bryant is a professor and associate chair of computer and information sciences at the University of Alabama at Birmingham (UAB), USA. He joined UAB in 1983 after completing his PhD in computer science at Northwestern University. His primary research focus is in theory and implementation of programming languages, and he has authored or coauthored more than 80 technical papers in these areas. Barrett is a member of ACM, the IEEE Computer Society, and the Alabama Academy of Science. He is an ACM distinguished lecturer and chair of the ACM Special Interest Group on Applied Computing (SIGAPP).

Carol C. Burt is president and CEO of 2AB (http://www.2ab.com), a specialist in trusted solutions for distributed business. Carol has been working in the development of software solutions for distributed computing for more than 25 years. Her technical background is in design and development of distributed computing middleware, software protocol converters, and security access control solutions. She holds leadership positions in the Object Management Group (OMG), as a member of the Board and the Architecture Board, and participates in distributed component technology research as an adjunct research professor at the University of Alabama at Birmingham (UAB). Carol holds a BS in mathematics and an MS in computer science.

Humberto Cervantes (humberto.cervantes@imag.fr) is a guest researcher in the software engineering group of the software systems and network research laboratory (LSR) of Grenoble University, France. His research focuses on supporting dynamic availability in component models. Other research interests include service orientation and dynamic reconfiguration. He received a PhD in computer science from the University Joseph Fourier in Grenoble, France.

Constantinos Constantinides is a lecturer at the School of Computer Science and Information Systems at Birkeck, University of London, UK. Prior to coming to Birkbeck, he was a visiting assistant professor at Loyola University Chicago. He also has taught at the Illinois Institute of Technology and Roosevelt University in Chicago. He holds a PhD in computer science from the Illinois Institute of Technology, an MS in computer science from the New York Institute of Technology, and a BSc in electronics from Keele University, UK. His research interests fall within the general areas of software engineering and programming languages, focusing on approaches that can support the

modularization of crosscutting concerns throughout software development, collectively known as aspect-oriented software development.

Steven A. Demurjian is a full professor and associate department head of computer science & engineering at the University of Connecticut with research interests of secure software design using UML and aspect-oriented programming, security requirements specification and assurance, RBAC/MAC models and security solutions for distributed environments and for XML documents, and reusability and refactoring for component-based systems (UML and Java). Dr. Demurjian has more than 100 publications in the following categories: one book, one edited book, eight journal articles, 23 book chapters, and 72 refereed articles.

Remco M. Dijkman obtained an MSc at the University of Twente in 2001. After his graduation, he worked at Ordina, a large software consultancy firm. Here, he was involved in the design of business processes and software to support business processes. At the same time, he worked as a researcher in the same area at the Open University of the Netherlands. In 2002, he returned to the University of Twente to work on his PhD thesis. His research focuses on the architecture of distributed systems and, particularly, the development of a modeling technique for describing their behavior.

Thuong N. Doan is a PhD student of computer science and engineering at the University of Connecticut, USA with research interests of secure software design using UML, security requirements specification and assurance, RBAC/MAC models, and applying logics in security assurance analysis.

Marlon Dumas received a PhD in computer science from the University of Grenoble, France in 2000. Since then, he has taken successive positions as postdoctoral fellow and lecturer at the Queensland University of Technology, Brisbane, Australia. His research interests are in the areas of Web services and business process technologies. He regularly serves as program committee member for international forums in these and related areas. He is a member of the IEEE Computer Society. For more information, visit http://www.fit.qut.edu.au/~dumas.

Schahram Dustdar is associate professor at the Distributed Systems Group, Vienna University of Technology, Austria. Since 1999, he has worked as the cofounder and chief scientist of Caramba Labs Software AG (http://CarambaLabs.com) in Vienna, a venture capital cofunded software company focused on software for collaborative processes in teams. His research interests are service-oriented information systems, process-oriented information systems, and distributed and mobile collaboration. For more information, visit http://www.infosys.tuwien.ac.at/Staff/sd/.

Harald Gall is professor of software engineering at the University of Zurich, Department of Informatics, Switzerland. Prior to that, he was associate professor at the Vienna

University of Technology in the Distributed Systems Group (TUV). His research interests are in software engineering with focus on software architectures, reverse engineering, long-term soft-ware evolution, program families, as well as distributed and mobile collaboration processes. For more information, visit http://www.ifi.unizh.ch/~gall.

Hamada Ghenniwa is an assistant professor at the Department of Electrical and Computer Engineering, The University of Western Ontario, Canada. Dr. Ghenniwa's main research expertise includes computational intelligence with a specific focus on intelligent agents, cooperation and coordination theory, as well as their application to cooperative distributed systems, such as electronic business and commerce, enterprise integration, health care, real-time systems, and manufacturing. He is the head of Cooperative Distributed Systems Engineering group in Bell-Centre for Information Engineering at the University of Western Ontario. He has authored and co-authored several papers in refereed journals and conference proceedings as well as technical and industrial project reports. Dr. Ghenniwa is currently leading research and industrial projects concerned with integration in distributed information systems, business-to-business e-commerce, and multiagent systems for manufacturing control.

Christopher D. Gill is an assistant professor of computer science and engineering at Washington University in St. Louis, Missouri, USA. His research interests include combining real-time, fault-tolerance, and security properties in middleware, distributed real-time and embedded systems, and communication and coverage strategies in mobile ad hoc networks and sensor networks. Dr. Gill developed the Kokyu scheduling and dispatching framework and led development of the nORB small-footprint real-time object request broker, both at Washington University. He has more than 50 technical publications and an extensive service record in standards bodies, workshops, and conferences for distributed real-time and embedded computing.

Richard S. Hall (richard.hall@imag.fr) is a guest researcher in the software engineering group of the software systems and network research laboratory (LSR) of Grenoble University, France. His research focuses on component and service orientation and mechanisms to dynamically assemble applications at runtime. Other research interests include software deployment, which was the focus of his PhD thesis. He received a PhD in computer science from the University of Colorado, Boulder.

Radu Handorean received a BS in computer science from the Politehnica University in Bucharest, Romania in 1999. In 2000, he began his graduate studies at the Department of Computer Science at Washington University in St. Louis. In 2003, he received his MSc in computer science and he is currently pursuing the doctorate degree. His research focuses on service-oriented computing and its particular challenges when applied in a mobile ad hoc networking environment.

John Hutchinson is a research associate in the Computing Department at Lancaster University, UK. His current research interests are in process support for component- and service-oriented software engineering. His recent and current research collaborations include extended component architecture design and management (ECOADM) and CBSEnet. ECOADM was a European Union-sponsored project whose aim was to develop tools and methods to support the development of component-based systems. CBSEnet is an EU-sponsored initiative, which aims to create a European-wide forum for the exchange of information between researchers and developers working in CBSE.

Maria-Eugenia Iacob is a scientific researcher at Telematica Instituut, The Netherlands since 2000. She holds a PhD in mathematical analysis from the University Babes-Bolyai of Cluj-Napoca, Romania. She also worked for this university from 1990-2000 as an assistant and then associate professor in the Department of Computer Science. At Telematica Instituut she has carried out research in several projects in the areas of business and e-business process (re)engineering and of information systems architectures.

Marijn Janssen is an assistant professor in the field of information systems for public administration at the Information and Communication Technology group of the Faculty of Technology, Policy and Management at Delft University of Technology, The Netherlands. His research interests include business engineering, adaptive architectures, simulation, electronic intermediaries, and agent-mediated coordination. He has been a consultant for the Ministry of Justice and received a PhD in 2001 in information systems.

Gerald Kotonya is a senior lecturer in software engineering at Lancaster University, UK. He has more than 10 years experience in software engineering research and development. His current research interests are in component- and service-oriented software engineering. His recent research collaborations include extended component architecture design and management (ECOADM) and CBSEnet. ECOADM was a European Union-sponsored project whose aim was to develop tools and methods to support the development of component-based systems. CBSEnet is an EU sponsored initiative, which aims to create a European-wide forum for the exchange of information between researchers and developers working in CBSE.

Jaroslav Král graduated at Faculty of Mathematics and Physics of the Charles University, Prague, Czech Republic in 1959. He has been working in computer science at Czech Academy of Sciences and several Czech universities. He is now a full professor at the Faculty of Mathematical Physics of Charles University Prague and a visiting professor at the Faculty of Informatics of Masaryk University Brno, Czech Republic. His current research interests include theory of formal languages and compilers, service-oriented systems engineering, and education of software experts. He has published more than 120 scientific papers. Jaroslav Král took part as the project leader in several successful projects including compilers, flexible manufacturing, and automated warehouse systems.

Marc M. Lankhorst started his research career in applications of genetic algorithms at the University of Groningen. He then moved to the Telematica Instituut where he became involved in modeling-related projects, for example, on business process modeling and analysis, modeling tools for middleware, and services for 2G/3G mobile networks and e-business modeling and architectures. Within these projects, he has held several research and management positions. Furthermore, he has been responsible for the Telematica Instituut's expertise management on modeling and architecture. Presently, he heads the group of application engineers of the institute and is project manager of a large multiparty project on modeling, visualization, and analysis of enterprise architectures. He teaches a course on reference models for networked applications at the University of Twente.

Steve Latchem is vice president of global business development and professional services at Select Business Solutions Inc., UK. Steve has been within the IT industry for more than 20 years, holding positions in large consultancy groups and IT departments ranging from business analyst to object-oriented consultant, architect, and project manager. Steve now directs the consulting group at SBS, which specializes in helping organizations to analyze, model, and develop high quality component-based solutions within multiple tier distributed architectures, using his experience of multiple object and component technology projects over the last 10 years. In addition, Steve has edited, collaborated, and co-authored books on software processes, design patterns, and component-based software engineering.

Yinsheng Li is an associate professor of School of Software, Fudan University, China. His current research interests include software agents and Web services and their applications in e-business and enterprise application integration. Dr. Li joined National Research Council of Canada and University of Western Ontario as Postdoctoral Fellow from 2001 to 2003. He received his PhD from Tsinghua University in 2001, his MSc from Southeast University in 1995, and his BSc from Chongqing University in 1992. He was working as project manager at Information Centre of National Building Material Industry Bureau from 1995-1997.

Zakaria Maamar, Zayed University, United Arab Emirates. In addition to receiving his PhD in 1998, Zakaria Maamar also received an MSc and BS in computer science from Laval University (Canada) and the Institut National d'Informatique, Algeria. Prior to joining Zayed University, he held a defense scientist position with the Defense Research Establishment Valcartier, Canada and an adjunct professor position in the Department of Computer Science of Laval University. His research interests lie in the areas of mobile computing, Web and mobile services, and software agents. His research work has been published in several international journals and magazines, such as *Communications of the ACM, Information & Software Technology* (Elsevier), and *Information Technology and Management* (Kluwer).

Eila Niemelä is a research professor of embedded software engineering at VTT Technical Research Centre of Finland and a docent of the University of Oulu. She obtained an MSc

and PhD in information processing science from the University of Oulu. Before graduation, she worked 15 years as a software engineer of embedded systems and from 1995-1998 as a senior research scientist in the embedded software research area at VTT. In 1998-2002, she led the Software Architectures Group at VTT. Architecture design, quality analysis on the architecture level, and service architectures of pervasive computing environments are her current research topics. She has published more than 50 scientific papers about component-based software architectures and middleware services. She is a member of IEEE and ACM.

Andrew M. Olson is professor emeritus of computer and information science at Indiana University Purdue University in Indianapolis, USA and adjunct research professor of computer science and software engineering at Butler University in Indianapolis. His research has ranged from symbolic-numeric computation through visual programming languages, human/machine interaction to, currently, software engineering of distributed computing systems with service-oriented architectures. Prior to these Indianapolis appointments, he served on the faculty of the Mathematics Department at the University of Puerto Rico, Rio Piedras, becoming professor and chair. Andrew spent four years in the Engineering Mathematics Department at the University of Chile, Santiago, before this.

Srinivas Padmanabhuni is currently a senior research associate at Software Engineering and Technology Labs in Infosys Technologies Limited, Bangalore, India. Dr. Srinivas specializes in Web services, service-oriented architecture, and grid technologies alongside pursuing interests in semantic Web, intelligent agents, and enterprise architecture. He has authored several papers in international conferences. Prior to Infosys, Dr. Srinivas has worked in multiple capacities in start-ups out of Canada and USA. Dr. Srinivas holds a PhD in computing science from University of Alberta, Edmonton, Canada. Prior to the PhD, he secured his BTech and MTech from Indian Institutes of Technology at Kanpur and Mumbai, respectively.

Charles E. Phillips, Jr. is a lieutenant colonel (LTC) in the US Army, holds a PhD in computer science and engineering from the University of Connecticut, and is currently an assistant professor with the Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, USA. In 23 years of service, LTC Phillips has held leadership positions of increasing responsibility as a communications and automation officer. His research areas of interest are RBAC/MAC models, security solutions for distributed environments, and information assurance in coalition warfare. LTC Phillips has 10 publications: three book chapters and seven refereed articles.

Giacomo Piccinelli is a research fellow in the Software Systems Engineering Group at the University College London (UCL), UK. His research focus is on service-oriented computing (SoC) and electronic services systems (ESS). Currently, he is scientific and technical coordinator for FRESCO (Foundational Research in Service Composition) and EGSO (European Grid for Solar Observation). Previously a member of technical staff at

HP Labs, he has been involved in the development of HP's e-service model, as well as other initiatives in the area of Web services. Results from his work contributed to HP's Service Composer and were recently discussed in the context of OMG and W3C workshops.

David Piper, principal consultant, Select Business Solutions Inc., UK, has worked in the IT industry since graduating from City University, London in 1979. Experience ranges from working with engineering companies supplying the automotive industry, extensive involvement with financial services companies, and as a consultant with the Select Business Solutions division of SBS. Working with SBS, David has helped many organizations to adopt component-based development, using the Select Perspective, and to customize the method in the light of project delivery experience. David's involvement has included working with companies of different sizes, system integrators and consultancies, and public sector bodies including government departments in the UK and overseas.

Rajeev R. Raje is an associate professor in the Department of Computer and Information Science at Indiana University, Purdue University Indianapolis, USA. Rajeev holds degrees from the University of Bombay (BE) and Syracuse University (MS and PhD). His research interests are in distributed-object computing, component- and service-based systems, and software engineering. Dr. Raje's current and past research has been supported by the US Office of Naval Research, National Science Foundation, Microsoft Corporation, and Eli Lilly and Company. Rajeev has published extensively in journals and conferences. He is also a member of ACM and IEEE.

Gruia-Catalin Roman received a PhD in computer science from the University of Pennsylvania in 1976. In the same year, he joined the faculty of the Department of Computer Science at Washington University in St. Louis. Roman is a professor and chairman of the department. He has an established research career with numerous published papers in a multiplicity of computer science areas including mobile computing, formal design methods, visualization, requirements and design methodologies for distributed systems, interactive high speed computer vision algorithms, formal languages, biomedical simulation, computer graphics, and distributed databases. Roman is an associate editor for ACM TOSEM and will serve as general chair for ICSE 2005.

George Roussos holds a BSc in mathematics from the University of Athens, an MSc in numerical analysis and computing from UMIST, and a PhD from Imperial College, London. Before joining the School of Computer Science and Information Systems at Birkbeck as a lecturer, he was the research and development manager for Pouliadis Associates Corporation, Athens, where he was responsible for the strategic development of new IT products in the areas of knowledge management and the mobile Internet. Previously, he was a Marie Curie research fellow at the Department of Mathematics, Imperial College, London, and then the Internet systems security officer for the Ministry of Defence, Athens. Dr Roussos has several years of experience in managing software

development teams in industrial, military, and academic settings. He is a member of the ACM and an associate of the IEEE and the IEEE Computer Society.

Roman Schmidt is a PhD student at the Swiss Federal Institute of Technology Lausanne (EPFL). He received an MSc in computer science from the Vienna University of Technology. His main research interests include the self-organization of distributed information systems, peer-to-peer systems, and semantic Web services.

Rohan Sen received a BS in computer science from Washington University in St. Louis, USA (2003). He began his graduate studies at the Department of Computer Science at Washington University in the Fall of 2003 and is currently working toward a doctorate in computer science. His research focuses on algorithms and middleware supporting service-oriented computing in ad hoc wireless settings.

Weiming Shen is a senior research scientist at the National Research Council, Canada's Integrated Manufacturing Technologies Institute. He received his BS in 1983 and MS in 1986, both degrees from Northern Jiaotong University, China, and his PhD degree in 1996 from the University of Technology of Compiègne, France. He has been working on intelligent agents and their applications to concurrent engineering design, intelligent manufacturing, and virtual enterprises for about 12 years. He has published one book, about 170 papers in scientific journals and international conferences/workshops, and coedited nine conference/workshop proceedings in the related areas. He is an editorial board member of the International Journal of Networking and Virtual Organizations and served as guest editor for another four international journals. He is senior member of IEEE and a member of AAAI, ACM, and ASME. He is also adjunct professor at the University of Western Ontario.

James Skene is a postgraduate researcher in the Computer Science Department of University College London, UK, having his BS from the same institution in 2000. His research interests lie in the areas of modeling languages, distributed systems, and performance analysis. He is primarily occupied with the TAPAS project, an IST framework 5 project addressing the problem of providing trustworthy electronic services with reliable performance properties across organizational boundaries. James is pursuing a related PhD, focused on the specification, management, and analysis of performance information in system specification.

Maarten Steen joined Telematica Instituut, The Netherlands in 1999. Dr. ir. Maarten Steen has been working on various topics related to e-business engineering, both in research projects as in applied projects with industry. His main research interests are modeling techniques, methods, and architectures for e-business applications. He has lectured on these topics at industry seminars and at universities. Before joining Telematica Instituut, Maarten Steen worked at the University of Kent at Canterbury on the application of formal methods in the area of open distributed processing. More specifically, he worked

on techniques for partial specification, such as consistency checking and composition and on enterprise modeling and policy specification.

Patrick Strating joined the Telematica Instituut, The Netherlands, in 1999. He has been working on various topics related to business networks. His main research interests are business architectures in the context of open business networks and enterprise modeling. He has given courses in business process innovation, business process modeling, and engineering, quantitative analysis, and Web service technology. Before joining Telematica Instituut, Patrick Strating worked in distributed and high performance computing at the Numerical Mathematics Department of the University of Twente.

Mahesh Subramanium is a graduate student at Oregon State University, Corvallis, Oregon, USA, registered to complete a master's degree in computer science. He has a Bachelor's of Technology in computer engineering from Kannur University, Kerala, India. He has more than two years of professional research and development experience. His main research interests are in the areas of information security, cryptography, intrusion detection and prevention, mobile intelligent agents, Web services and distributed systems.

Hugo ter Doest is a scientific researcher at the Telematica Instituut, The Netherlands, a public/private research consortium consisting of academia and companies. His main focus is on architecture modeling and design ranging from domain-specific architectures to enterprise architectures. He has been involved in several research and consultancy projects with an emphasis on development and application of methods and techniques for architecture description, implementation, and management. Currently, he is responsible for the adoption of ArchiMate project research results by market parties.

Richard Yi Ren Wu obtained a Master's of Computer Science from the University of Alberta, Canada and is a senior technical consultant of Marlborough-Stirling Group offering software products and outsourced services in the international financial services sector. Having worked in the software industry throughout his career, Richard has expertise in software architectures and engineering, project management, and mobile distributed computing as well as public institutional education. His over 15 years of experience includes commercial wholesales and distribution systems for IBM Canada, mobile call processing systems for BCTel Wireless Mobility, financial investment and data warehousing systems for Hong Kong Bank of Canada, international language grammar checkers for Microsoft Corporation, XML document distributed authoring systems for SoftQuad, online payment transactions and financial/vendor systems for Amazon.com, and Internet-based lending and ASP-based brokerage systems for Marlborough-Stirling Canada.

Michal Žemlička is an assistant professor at the Faculty of Mathematics and Physics of Charles University, Prague. He graduated in 1996. His current research interests are

extensible compilers, theory of parsing, the design of large software systems, data structures, and computational linguistics. He has published more than 20 scientific papers.

Jiehan Zhou received a PhD in manufacturing and automation from Huazhong University of Science and Technology (Wuhan, PR China) in 2000. He is being sponsored by the ERCIM (European Research Consortium for Informatics and Mathematics) fellowship for studying agile software development method in VTT (Oulu, Finland) and semantic Web in INRIA (France). Before that, Dr. Zhou carried out two years postdoctoral research in Computer Integrated Manufacturing System in National CIMS Center of Tshinghua University (Beijing, PR China). He has published several papers on manufacturing engineering, software engineering, and knowledge engineering.

Index

O

P

Q

R

S

T

U

V

W

X