ADVANCES IN INTELLIGENT AND SOFT COMPUTING 97

Wojciech Zamojski
Janusz Kacprzyk
Jacek Mazurkiewicz
Jarosław Sugier
Tomasz Walkowiak (Eds.)

Dependable Computer Systems

Springer

Advances in Intelligent and Soft Computing 97

Editor-in-Chief: J. Kacprzyk

Advances in Intelligent and Soft Computing

Editor-in-Chief

Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 87. E. Corchado, V. Snášel, J. Sedano, A.E. Hassanien, J.L. Calvo, and D. Ślęzak (Eds.)
Soft Computing Models in Industrial and Environmental Applications, 6th International Workshop SOCO 2011
ISBN 978-3-642-19643-0

Vol. 88. Y. Demazeau, M. Pěchoucěk, J.M. Corchado, and J.B. Pérez (Eds.)
Advances on Practical Applications of Agents and Multiagent Systems, 2011
ISBN 978-3-642-19874-8

Vol. 89. J.B. Pérez, J.M. Corchado, M.N. Moreno, V. Julián, P. Mathieu, J. Canada-Bago, A. Ortega, and A.F. Caballero (Eds.)
Highlights in Practical Applications of Agents and Multiagent Systems, 2011
ISBN 978-3-642-19916-5

Vol. 90. J.M. Corchado, J.B. Pérez, K. Hallenborg, P. Golinska, and R. Corchuelo (Eds.)
Trends in Practical Applications of Agents and Multiagent Systems, 2011
ISBN 978-3-642-19930-1

Vol. 91. A. Abraham, J.M. Corchado, S.R. González, J.F. de Paz Santana (Eds.)
International Symposium on Distributed Computing and Artificial Intelligence, 2011
ISBN 978-3-642-19933-2

Vol. 92. P. Novais, D. Preuveneers, and J.M. Corchado (Eds.)
Ambient Intelligence - Software and Applications, 2011
ISBN 978-3-642-19936-3

Vol. 93. M.P. Rocha, J.M. Corchado, F. Fernández-Riverola, and A. Valencia (Eds.)
5th International Conference on Practical Applications of Computational Biology & Bioinformatics 6-8th, 2011
ISBN 978-3-642-19913-4

Vol. 94. J.M. Molina, J.R. Casar Corredera, M.F. Cátedra Pérez, J. Ortega-García, and A.M. Bernardos Barbolla (Eds.)
User-Centric Technologies and Applications, 2011
ISBN 978-3-642-19907-3

Vol. 95. Robert Burduk, Marek Kurzyński, Michał Woźniak, and Andrzej Żołnierek (Eds.)
Computer Recognition Systems 4, 2011
ISBN 978-3-642-20319-0

Vol. 96. A. Gaspar-Cunha, R. Takahashi, G. Schaefer, and L. Costa (Eds.)
Soft Computing in Industrial Applications, 2011
ISBN 978-3-642-20504-0

Vol. 97. W. Zamojski, J. Kacprzyk, J. Mazurkiewicz, J. Sugier, and T. Walkowiak (Eds.)
Dependable Computer Systems, 2011
ISBN 978-3-642-21392-2

Wojciech Zamojski, Janusz Kacprzyk,
Jacek Mazurkiewicz, Jarosław Sugier,
and Tomasz Walkowiak (Eds.)

Dependable Computer Systems

Springer

Editors

Wojciech Zamojski
Wrocław University of Technology
Institute of Computer Engineering,
Control and Robotics
ul. Janiszewskiego 11/17
50-372 Wrocław
Poland
E-mail: wojciech.zamojski@pwr.wroc.pl

Janusz Kacprzyk
Polish Academy of Sciences
Systems Research Institute
ul. Newelska 6
01-447 Warszawa
Poland
E-mail: kacprzyk@ibspan.waw.pl

Jacek Mazurkiewicz
Wrocław University of Technology
Institute of Computer Engineering,
Control and Robotics
ul. Janiszewskiego 11/17
50-372 Wrocław
Poland
E-mail: jacek.mazurkiewicz@pwr.wroc.pl

Jarosław Sugier
Wrocław University of Technology
Institute of Computer Engineering,
Control and Robotics
ul. Janiszewskiego 11/17
50-372 Wrocław
Poland
E-mail: jaroslaw.sugier@pwr.wroc.pl

Tomasz Walkowiak
Wrocław University of Technology
Institute of Computer Engineering,
Control and Robotics
ul. Janiszewskiego 11/17
50-372 Wrocław
Poland
E-mail: tomasz.walkowiak@pwr.wroc.pl

ISBN 978-3-642-21392-2 e-ISBN 978-3-642-21393-9

DOI 10.1007/978-3-642-21393-9

Advances in Intelligent and Soft Computing ISSN 1867-5662

Library of Congress Control Number: 2011928737

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India

Printed on acid-free paper
5 4 3 2 1 0
springer.com

Preface

We would like to present the monographic studies on selected problems of dependable computer systems and networks which are published in the series *Advances in Intelligent and Soft Computing*.

The systems under consideration

Contemporary systems are created as very sophisticated products of human ideas and they are characterized by complex structure. The three main elements that should be identified in any system are: users, services (functionalities), and technological resources. The technological resources are understood as technical assets (engineering stuff) and information resources (algorithms, processes and management procedures). In the most general operation flow the users generate tasks which are realized by the system. The task to be realized requires some services (functionalities) available in the system and realization of the services needs a defined set of technical resources. In a case when any resource component of this set is in the state "out of order" or "busy", the task may wait until the moment when the component returns to the state "available", or the service may try to create other configuration based on available technical resources.

The modern systems are equipped with suitable measures which minimise the negative effects of these inefficiencies (a check-diagnostic infrastructure, fault recovery, information renewal, time and hardware redundancy, reconfiguration or graceful degradation, restart etc). The special service resources (service persons, different redundancy devices, etc.) supported by so called maintenance policies (procedures of resource services used to minimize negative consequences of faults that are prepared before or created ad hoc by the system manager) are incorporated in every real system.

System dependability

The dependability is a modern approach to reliability problems of contemporary systems. It is worth to underline the difference between the terms of system dependability and systems reliability. Dependability of systems, specially computer systems and networks, is based on a multi-disciplinary approach to theory, technology, and maintenance of systems operating in real (and very often unfriendly) environment. Dependability of systems concentrates on a probability of tasks realization by a system which is the unity of technical, information and human resources, while "classical" reliability focuses mainly on technical system resources.

The system dependability can be described by such attributes as availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences for the users and the environment), security (availability of the system only for authorized users), confidentiality (absence of unauthorized disclosure of information), integrity (absence of improper system state alterations) and maintainability (ability to undergo repairs and modifications).

System dependability and soft computing

The following main assumptions are usually made during dependability analyses and syntheses: system user tasks are realized on the basis of available services (functionalities) and information or technical resources. This means that the realized task is dynamical mapped on the services and then the services are dynamical mapped on the system resources. The system operates in unfriendly environment and its components (services and resources) are working with limited performance and with unreliable parameters, so consequently the user task is executed with limited performability parameters too. Sometimes a combination of unfriendly conditions of the environment together with possible faults produced by the technical infrastructure and/or by the users may create a critical situation in system operation which may lead not only to incorrect task realization but even to system collapse.

The functional and reliability parameters of system functionalities (services) and resources are often tightly inter-dependent – hence dependability analysis or synthesis of such contemporary systems need adequate formal and mathematical models and calculation (evaluation) methods suitable for systems and processes which are created by mix of stochastic and deterministic events generated by hardware resources, information resources (algorithms and procedures of operations and system management) and human-factors (managers, administrators and users). It is very often difficult to find a relation between system elements and system events (the relation between reasons and results) and it is even more difficult to define mathematical models with "analytical" relationships between such phenomena as, for example, a system user (administrator) mistake and the time extension of task execution in a distant node of the system. Of course, these problems are not only associated with human factors but the same difficulties are generated by complexity of technological and information system resources too. The problems may be solved only by using artificial intelligence and soft computing methods and, in this situation, contemporary systems, especially computer systems and networks, are the actual examples of undependable artificial intelligence systems which are not formally and mathematically described yet.

The topical scope of the monograph

In the following few points we are presenting the main subject areas of the chapters selected for our monograph.

Methodology

Compliant methodology and tools to develop and manage development environments of IT security-enhanced products and systems for the purposes of their certification are standardized in the ISO/IEC 15408 Common Criteria. In chapter 1,

Białas presents results of research on how to develop the set of patterns for different kinds of evidences that should be delivered together with the IT product or system for independent evaluation of its dependability, safety, security etc. levels. The work summarizes the methodology with respect to the achieved and planned project results.

An original methodology for dependability analysis which uses semi-Markov models is described in chapter 16. In this approach the models represent equipment ageing and also incorporate various maintenance activities. Having available some basic representation it is possible to adjust its parameters so that it corresponds to some hypothetical new maintenance policy and then to examine impact that this new policy has on various reliability characteristics of the system. The work deals with a methodology of model adjustment and specifically investigates its one particular problem: avoiding probability saturation in cases when the tuning is aimed at reaching increased repair frequencies.

Mathematical models

A model of client – server computing system is proposed by Zuberek in chapter 23. The main problem is how to model services requested by clients. The author presents a component language for description of system elements (service, client) and proposes interleaving requests from different components to increment performance of the system.

Design models can be analyzed to predict whether the future system satisfies requirements prior to its implementation. In Ghezzi and Sharifloo's work (chapter 4) it is proposed to use a model-driven approach in analyzing design models against non-functional requirements (reliability, performance, cost, and energy consumption). The authors also show how probabilistic model checking techniques can be used to achieve this purpose. In the method it is assumed that initially the software engineer describes the desired system functionalities using behavioural models that represent high-level scenarios specified by UML Sequence Diagrams.

Functional approach to dependable operation of the system considered as network of services is presented by Walkowiak and Michalska in chapter 21 and by Toporkov et al. in chapter 19. The latter work proposes a slot selection algorithm with non-rectangular time-slot window. The slot algorithms are used in economic models for independent job batch scheduling in distributed computing with non-dedicated resources. Economic models of scheduling are based on the concept of fair resource distribution between users and owners of computational nodes. They are effectively used in such spheres of distributed computing as Grid, cloud computing, and multiagent systems. Resource brokers usually implement some economic policy in accordance with the application-level scheduling concept. When establishing virtual organizations, the optimization is performed for the job-flow scheduling. Application of the set of specific rules leads to overall increase in the quality of service (QoS) and in efficiency of resource usage.

Duration graphs are an extension of timed automata and are suitable for modelling the accumulated times spent by computations in duration systems. Majdoub in chapter 12 proposes a framework for automatic generation of test cases directly from the specification model represented by states and trees.

Software

Nowadays computer systems fail mainly due to software faults. One of the main reasons of software failures is software ageing. The ageing is the progressive software performance degradation due to accumulation of error conditions that leads to system resource exhaustion. To counteract aging, software rejuvenation has been recently proposed. Rejuvenation is the concept of periodically stopping the software, cleaning its internal state and then restarting. Koutras in chapter 8 examines how software availability is affected by rejuvenation technology and tries to find an optimal rejuvenation policy in terms of availability and downtime cost.

Effective testing is one of the key issues in development of dependable software systems. The objective of Bluemke and Kulesza's work in chapter 2 is to compare dataflow and mutation testing of several Java programs. Results of experiments conducted in the Eclipse environment are also included.

Computer oriented languages are very often used in system modelling and the examples can be found in chapters 2, 4, 9, 10, 18 and 13. The UML formalization is often undertaken by projecting the notation in a rigorously defined semantic domain. When the target formalism is of state transition type, the derived models are verified by model checking. Meziani and Bouabana-Tebibel verify in chapter 13 a model using the Petri nets approach. The work of Kowalski and Magott in chapters 9 and 10 proposes UML models of maintenance policies which are built on the base of fault trees and Petri nets. The problem of developing dependable software (a metamodel and an UML profile for functional programming languages) is discussed by Szlenk in chapter 18.

Tools and technologies for dependable system operation

In many applications microcontroller circuits are used for improving dependability of reactive systems with real time requirements. In order to evaluate their fault susceptibility various fault injection techniques have been developed. The software for fault injection analysis applied for a case of satellite microcontroller onboard power system is presented by Iwiński and Sosnowski in chapter 5.

Memory failures are quite common in contemporary technology. When they occur, the whole memory bank has to be replaced, even if only few bytes of memory are faulty. With increasing sizes of memory chips the urge not to waste these "not quite properly working" pieces of equipment becomes larger and larger. Operating systems such as Linux already provide mechanisms for memory management which could be utilized to avoid allocating bad memory blocks which have been identified earlier, allowing for a failure-free software operation despite hardware problems. Surmacz and Zawistowski describe in chapter 17 problems of detecting memory failures and deal with OS (Linux) mechanisms that can be used for bad block exclusion.

The crucial role of evaluation during development of the information retrieval systems is to provide useful evidence of performance improvements and the quality of results that they return. However, the classic evaluation approaches have limitations and shortcomings especially regarding the user consideration, the measure of the adequacy between the query and the returned documents and the consideration of characteristics, specifications and behaviours of the search tool.

Bouramoul et al.'s chapter 3 presents a new approach for Web search engines evaluation that takes into account the context during the assessment process. The experiments included at the end prove the applicability of the proposed approach to the real research tools.

Theory of the systems reliability is particularly applicable to electronic protection systems (alarm systems) which, due to their specific character of use, should be characterized by the high level of reliability. The devices and electronic units applied in the wide range in those systems, in particular the microprocessor systems, require a new perspective on the system reliability and safety. The Sergiejczyk and Rosinski's chapter (15) presents a reliability analysis of the electronic protection systems using optical links. Some theoretical and practical problems of detecting encrypted files in evidence data during digital forensics investigations are presented by Jóźwiak et al. in chapter 6.

Probabilistic assessment methods

One of the most difficult problem in development of the critical computer based systems is evaluation of system measures that are strongly related to probabilistic description of the events in the considered system. Sharvia and Papadopoulos (chapter 14) propose an approach to safety analysis which systematically integrates both compositional and behavioural safety analysis paradigms. The process starts with compositional analysis and uses its results to provide a systematic construction and refinement of state machines, which can be subsequently analyzed by behavioural analysis. Kharchenko et al. in chapter 7 try to built a technique for assessment of probabilistic dependability metrics of systems with different structures and failures. The technique is useful for synthesis of systems with applied redundancy and FTC concepts.

Summary

The brief overview of the chapters above illustrates a wide diversity of dependability problems in contemporary technical systems. We believe that this monograph will be interesting to all scientists, students, practitioners and researchers who deal with problems of dependability on practical grounds. It is our hope that it may be an inspiring source of original ideas and can help to define new challenges, as well as can provide a general insight into selected topics of the subject. As the editors we would like to express our sincere gratitude to all authors who have contributed to this volume and to all reviewers whose remarks has helped us to refine its contents.

Wojciech Zamojski
Janusz Kacprzyk
Jacek Mazurkiewicz
Jarosław Sugier
Tomasz Walkowiak

Contents

Patterns Improving the Common Criteria Compliant IT Security Development Process

Andrzej Białas

Institute of Innovative Technologies EMAG,
40-189 Katowice, Leopolda 31, Poland
e-mail: a.bialas@emag.pl

Abstract. The chapter concerns the project of the methodology used to create and manage development environments of IT security-enhanced products and systems for the purposes of their future Common Criteria certification. The key issues of the patterns-based project are discussed: how to develop the set of patterns for different kinds of evidences to be delivered with the IT product or system for independent evaluation. The author characterizes the IT security development process and the elaborated evidences, and presents analyses provided to develop such patterns. The patterns usage is shown by a few examples which are part of a more complex case study. Such patterns facilitate and speed up the IT security development process, improve the quality of evaluation evidences, as they are more consistent and include all details required by the considered assurance components, facilitate the computer support of the IT security development process. The chapter concludes the methodology with respect to the achieved and planned project results.

1 Introduction

Information technologies (IT/ICT) are often meant to fulfil social and business objectives in high-risk environments. The main issue, related to the term "assurance", is which factors ensure that the measures meant to protect certain resources in a critical situation will really work. The assurance is understood that an IT product or system meets its security objectives expressing these measures. The basic assurance methodology is specified within the ISO/IEC 15408 Common Criteria (CC) standard [14]. According to the CC paradigm the source of measurable assurance are the rigour applied during the development and manufacturing processes, independent third-party evaluation, operation and maintenance according to the received certificate. The reliable IT products and systems with measurable level of assurance are developed in a rigorous manner in special "development environments" (exactly: development, production and maintenance environments). Assurance is measured with the use of Evaluation Assurance Levels (EALs) in the range: EAL1 (min.) to EAL7 (max.).

The chapter presents the concept, first results and future plans of the CCMODE (Common Criteria compliant, Modular, Open IT security Development Environment)

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 1–16.
springerlink.com

R&D Project carried out by the Institute of Innovative Technologies EMAG [13]. CCMODE is co-financed by the EU within the European Fund of Regional Development. The objective of the project is to work out a CC-compliant methodology and tools to develop and manage development environments of IT security-enhanced products and systems for the purposes of their future certification. The basic products of the project will be the following: knowledge, patterns (including documentation, procedures, evidences, etc.), methodology and tools used to create and manage development environments by different business organizations. The IT products and systems developed in these environments, having measurable assurance (EAL), can be certified and used in high risk applications.

CCMODE assumes participation in activities dealing with the Common Criteria methodology improvement and extending the range of the standard application:

1. The project does not provide a solution dedicated to one environment only. It provides patterns and methods for developing a wide range of environments – generally where the Common Criteria standard is applicable. For this purpose a set of modules will be developed, along with the methodology of adapting them to the needs of a given environment. The possible reusing of the same patterns many times will bring significant financial benefits.
2. The project will provide an integrated solution (based on ISO/IEC 15408 and ISO/IEC 27001) for the IT security management system, dedicated to development environments. The solution will enable better protection of information related to projects carried out in these environments and will consider business continuity aspects of manufacturing and maintenance processes.
3. The project considers both – traditional "sites" organized to develop a certain IT product or system, and sites compliant with the "site-certification" concept [24], allowing to reuse some evidences in the elaboration of a certain group of similar IT products or systems.
4. The project will provide a computer tool which will support even the most laborious and difficult operations related to the management of the environment and product or system in the life cycle (security analyses, management of evidences, configuration, flaws and tools). This will bring extra benefits to organizations in the form of management processes automation.
5. The project is strongly based on the life cycle model, knowledge engineering and risk analysis methods.

The project has its national aspect – disseminating and finally implementing the ISO/IEC 15408 Common Criteria (CC) standard in Poland, creating the CC community and elaborating best practices.

The chapter gives a short introduction to the project domain, surveys current state of the works, discusses the key issues of the project: how to develop the set of patterns for the security specification means and for the evaluation evidences to be delivered with the IT product or system for independent evaluation. All kinds of patterns are identified and presented. Some of them are shown by examples. The last section concludes the project efforts and specifies the planned works.

2 Project Background

The project background is founded on the Common Criteria methodology specified within the [14]. The project concerns the development-, production- and maintenance processes of any IT product or system which should be protected against threats. In practice the Common Criteria standard, as well as the CCMODE project, concerns all kinds of IT products or systems, called TOEs (Targets of Evaluation). The TOE can be: hardware, software, combination of the above, application programs, tools for products development, complete systems, etc. More than 1200 TOEs have been developed and evaluated so far [15].

2.1 The Common Criteria IT Security Development Methodology

The Common Criteria methodology [14] encompasses the three main processes:

- the IT security development process, related to the elaboration of the document called ST (Security Target) or PP (Protection Profile) for the given TOE, presenting the TOE, specifying its security problem definition, security objectives, requirements, and finally the TOE security functions for ST which are later implemented on the claimed EAL level;
- the TOE development process, related to the IT product or system development with the use of the assumed technology, including its TOE security functions implementation on the claimed EAL level;
- the IT security evaluation process, performed by an independent, accredited security lab – the process completed by the certification; during this process the TOE, its Security Target and evidences are evaluated according to the CC methodology with respect to the declared EAL requirements.

The Common Criteria (CC) methodology is matured but still improved. The main challenges are: raising the design preciseness, facilitating the development and evaluation, decreasing the development cost and time. The general motivation of the author's works is to improve the IT security development process using the advantages and new possibilities offered by the patterns-based approach, and to minimize the barrier for developers related to the lack of knowledge, methods and exemplars of evidences, etc.

Thanks to the patterns, the evidences can be elaborated in a unified proven way and developers can acquire knowledge how to elaborate evidences. More information about the Common Criteria methodology can be found in [15], [18], [9], [2].

2.2 Patterns-Based Development

Design patterns are often used by engineers in many technology domains, including here discussed IT- and IT security domains. These patterns can be considered as reusable, proven solutions to problems with respect to a specific context. They

provide development process knowledge helping to achieve the expected solution within the project domain. Patterns-based development enforces standard solutions, saves time and money and improves quality.

The patterns are specified in a formalized way, e.g. using UML (Unified Modelling Language), OCL (Object Constraints Language), different kinds of codes, ontologies, formalized descriptors, etc. Numerous patterns concern software architecture, including applied security mechanisms.

The paper [25] surveys approaches to the security patterns categorized with respect to the software life cycle phases, i.e. requirements-, design- and implementation phases. No approaches closely related to the Common Criteria methodology were encountered in this paper.

The commonly used security design patterns related to software solutions are specified in [23]. These architectural patterns related to the enterprise management applications concern, for example: enterprise security and risk management, identification and authentication, access control models and systems, operating system access control, accounting facilities, firewall architecture, secure Internet applications, IP telephony, and cryptographic key management. With respect to the CC methodology, this kind of security design patterns can be used to implement the security functions (on a given EAL) within an IT product or system.

The book [19] introduces a UML extension called UMLsec. It provides a unified approach to security features description during the secure systems development. UMLsec allows to define the UML patterns that encapsulate the design knowledge in the form of recurring design problems, and consist of the "pattern name", "problem description", "problem solution", and "consequences". The established formal rules of security engineering can be used by a wider group of developers. The elaborated patterns present the basic security engineering solutions, like: electronic signature, secure Java programs, electronic purse, secure channel, TLS Internet protocol, bank applications, biometric authentication systems, etc. Please note that these patterns focus on modelling IT security features and behaviour within the system. The UMLsec patterns defined in [19] do not concern the IT security development process compliant with the Common Criteria standard, although they can be helpful in this process, e.g. to evaluate UML specifications against different vulnerabilities. This issue was discussed in the monograph [9].

The author's preliminary researches discussed in the Section 2.3 concern security specification means patterns. They are not focused on IT security functionality, as the above mentioned security design patterns, but rather on risk management- and assurance issues closely related to the Common Criteria methodology. From this point of view they differ from the above mentioned on one hand, and supplement them on the other hand. The security specification means patterns are related to the IT security development process mentioned in the Section 2.1.

CCMODE adopts the specification means patterns, though it is focused on the development of a new class of the CC-related patterns – patterns of the evaluation evidences, related mainly to the TOE development process. There are no researches

focused on the elaboration of evaluation evidences patterns. Some guidance is provided by the standard [14] and BSI guide [17] restricted to EAL 1-5.

2.3 The CCMODE Project Background and Current Results

The project background was based on the preliminary, multidirectional researches.

The first direction was focused on the UML/OCL modelling. The Common Criteria compliant, UML/OCL-based IT security development framework (ITSDF) [9], [3], [10], [8] was elaborated, which embraces:

- models of data structures and processes of IT security development stages, including: security problem definition (SPD), security objectives (SOs) elaboration, security functional requirements (SFRs), and TOE security functions (TSFs) workout; two kinds of classes are distinguished: classes responsible for the ST elaboration and classes representing specification data containers;
- models of the specification means used for these IT security development stages, including CC security components – functional- (SFRs) and assurance components (SARs) [14] and the semiformal enhanced generics; please note that generics and components constitute a security specification language.

The semiformal ITSDF framework was implemented as a software tool to support IT security developers. Please note that the introduced enhanced generics are derived from "generics" commonly used by IT security developers. The enhanced generics are defined as mnemonic names expressing common features, behaviours or actions related to IT security issues, like: subjects (active entities), objects (passive entities), threats, assumptions, organizational security policies (OSP), security objectives, and security functions [3], [9]. They are "enhanced" since they are semiformal and have features comparable to CC components, allowing such operations as: parameterization, derivation, iteration, and refinement. Enhanced generics can be grouped by domains of applications. They allow to create generics chains which express solutions to elementary security problems. This way the preliminary version of the Security Target (ST) pattern and specification means patterns used to fulfil this ST structure was elaborated.

The second direction of the preliminary researches deals with the application of the ontological approach [21], [22] to this framework [4], [7]. Please note that the ontology represents explicit formal specifications of the terms in the project domain and relations between these terms. The elaborated ITSDO (IT Security Development Ontology) represents the security requirements structures (i.e. ST, PP), specification means to fill in these structures with contents for different TOEs (author's defined enhanced generics, CC-defined functional and assurance components) as well as patterns for evidences. During these works about 350 enhanced generics were predefined as elementary items designed to specify general security features of commonly used IT products or systems.

The third direction of preliminary researches is focused on the validation of the specification means patterns on different TOE designs (firewall [12], motion

sensor [5], [6], medical sensor [2], gas detecting sensor [1]), their improvement and the elaboration of patterns for selected evidences.

CCMODE should bring these partial results together, supplement them and create a complex and unified CC related patterns system facilitating the IT security development. To achieve this, the following 7 project tasks were scheduled:

1. Research of development environments of IT security-enhanced products with respect to life cycle processes – identification of developers' needs and expectations, building a reference model.
2. Working out an open, modular development (manufacturing, maintenance) environment – patterns developed for evidence material related to the environment and to the product developed in this environment.
3. Working out a methodology for implementation and management of development environments – key issues: how to adapt open patterns to the identified needs, how to develop an environment based on these needs, and how to manage this environment.
4. Validation of the implementation methodology – with the participation of security developers and evaluators.
5. Working out a tool to support the development environment management – automation of difficult and repeatable operations.
6. Validation of the tool supporting the management of development environments – with the participation of security developers and evaluators.
7. Creating an experimental development environment – also: promotion, CMODE community.

The task 1 focused on researches of development environments of IT security-enhanced products or systems with respect to the life cycle processes. The knowledge was acquired about the structure, operation of development environments of security-enhanced products. The assurance components were analyzed in terms of their relationships with the development environment and the IT product or system developed in the environment. On this basis the reference model of the development environment with an embedded life cycle was made. Then the method to assess the compliance of any development environment with the CC-based reference model was elaborated and validated with the participation of IT developers.

The task 2, closely related to this chapter, was focused on working out an open, modular development (manufacturing, maintenance) environment, represented by a set of Common Criteria related patterns, i.e.:

- SST (Site Security Target) document pattern for a local development environment according to the site certification approach,
- evidences patterns to evaluate the environment based on the requirements of the ALC (Life cycle support) assurance class,
- pattern modules of the information security management system compliant with ISO/IEC 27001 for the development environment,
- security specification patterns in the form of Security Target (ST) and Protection Profile (PP), including their low-assurance versions,
- evidences patterns for IT products and systems developed in the environment.

Currently the set of patterns of the evaluation evidences as well as the specification means are elaborated and the third task of the project has been carried out – focused on the implementation of the development environments. The gap analysis method has been elaborated. The organization which plans this implementation should be first audited to identify its needs, restrictions and any incompliance with the Common Criteria standard. The gap analysis gives input to the adaptation of patterns for the created development environment. The customized patterns define the development-, production- and maintenance processes of the organization, reflecting the assumed life cycle model and helping to manage this environment.

3 Patterns in the CCMODE Project

There are three kinds of patterns considered in the project: patterns of evaluation evidences, patterns of specification means, and patterns of auxiliary documents.

The patterns of evaluation evidences required by the standard and delivered for the independent evaluation are of key importance for the CCMODE project. They play the roles of "open modules" that should be refined and customized according to the developers' needs and expectations to implement the development environment for the given IT product or system, and to elaborate evaluation evidences for the given TOE. They will be discussed in the next subsection.

The patterns of the specification means include the author's defined enhanced generics and standard-defined functional and assurance components. They can be expressed informally using the dot separated notation, semi-formally using the UML notation or as knowledge base items. Currently the enhanced generics are validated on different designs and optimized. Moreover the generics subsets for particular domains of application are elaborated, e.g. for intelligent sensors [1], [2]. The patterns of the specification means are omitted in this chapter because the author's many publications have discussed this issue.

During the development-, manufacturing- and maintaining processes as well as the evaluation process many auxiliary documents, procedures, forms, checklists, reports, etc. can be useful. Many patterns belonging to this family are related to the ISMS (Information Security Management System) implementation within the development environment. These patterns will be omitted in this chapter.

3.1 Identification of the Evaluation Evidences Patterns

Two important issues were solved during the initial tasks of CCMODE: how many patterns of the evaluation evidences are needed and what their organization and shape look like.

Analyzing the contents of the assurance classes components, five groups of evaluation evidences patterns were distinguished:

- patterns for basic security requirements,
- patterns of evaluation evidences closely related to the development environment, its organization and development-, production- and maintenance processes,

- patterns of evaluation evidences closely related to the IT product or system (TOE),
- patterns of evaluation evidences related to both the development environment and the IT product or system,
- patterns of evaluation evidences related to the composition.

Basic security requirements are considered as evaluation evidences as well. The following five patterns were defined:

- Security Target pattern (STp), low assurance Security Target pattern (laSTp) used for EAL1 – both based on the ASE (Security Target Evaluation) assurance class,
- Protection Profile pattern (PPp), low assurance Protection Profile pattern (laPPp) used for EAL1 – both based on the APE (Protection Profile Evaluation) class,
- Site Security Target pattern (SSTp) – based on the AST (Site Security Target Evaluation) class defined outside the [14], i.e. in the [24] guide.

The first four of them contain basic security requirements for the TOE, while the latter one, concerning the site certification, deals with the site, considered as the development environment. Each Common Criteria project uses one of the basic security requirements. This main document helps to organize other evidences. Site Security Target is optional.

The second group of patterns is closely related to the requirements for the development environment, its organization and its development-, production- and maintenance processes. All these issues are specified in seven assurance families belonging to the ALC (Life-cycle support) assurance class:

- Life-cycle model definition pattern (ALC_LCDp) presents the high-level description of the TOE life-cycle and creates the framework for the entire development environment;
- Development security pattern (ALC_DVSp) presents physical, procedural, personnel, and other measures used in the development environment to protect the TOE and its parts; it can be extended by the Information Security Management System pattern (ISMSp) compliant with ISO/IEC 27001; ISMSp represents an extensive set of patterns needed to implement the standard in the development environment to better protect design data and manage information security;
- Configuration management (CM) capabilities pattern (ALC_CMCp) defines a more detailed description of the management of the configuration items and enforces discipline and control in the processes of refinement and modification of the TOE and the related information;
- Configuration management scope pattern (ALC_CMSp) shows how to specify items to be included as configuration items and hence controlled by the above CM capabilities (ALC_CMCp);
- Tools and techniques pattern (ALC_TATp) is responsible for control tools, their options and techniques used in the development environment (programming languages, documentation, implementation standards, runtime libraries, different equipment, etc.);

- Delivery pattern (ALC_DELp) describes the secure transfer of the finished TOE from the development environment into the responsibility of the user;
- Flaw remediation pattern (ALC_FLRp) presents requirements that the detected security flaws should be traced and corrected by the developer.

It was assumed that for each assurance family of the given assurance class only one pattern of evidences will be defined. Please note that in the given family there are many possible assurance components of rising rigour. This is considered in the pattern definition by extra sections or details. Each of the mentioned seven patterns was developed in two versions: in the basic, TOE-centric version (development environment is organized for the given TOE) and in the site-certification version (development environment as the "site" is organized for the certification and then to develop inside it a certain group of the TOE) – nuances of these two approaches are omitted in this chapter.

The third group of evaluation evidences patterns is closely related to the IT product or system (TOE). As it was mentioned above for the given assurance family only one pattern has been elaborated. Rising rigour represented by hierarchically ordered components is expressed by extra sections or details in patterns. All TOE-related issues are specified in 12 assurance families belonging to the ADV (Development), AGD (Guidance documents) and ATE (Tests) assurance classes:

- Security Architecture pattern (ADV_ARCp) presents a description of the security architecture of the TOE security functions to show if they achieve desired properties;
- Functional specification pattern (ADV_FSPp) describes the TOE security functions (TSFs) interfaces (TSFIs) which consist of all means for users to invoke a service from the TSF (by supplying data that is processed by the TSF) and the corresponding responses to those service invocations;
- TOE design pattern (ADV_TDSp) provides context for a description of the TSFs and describes the TSFs; with respect to the applied rigour (EAL) the TOE decomposition is specified on different levels of detail (subsystems, modules);
- Implementation representation pattern (ADV_IMPp) expresses how the TSFs are implemented (software/firmware/hardware design language source code, hardware/IC diagrams, layouts, binary files);
- TSF internals pattern (ADV_INTp) addresses the assessment of the internal structure of the TSFs; the well-structured TSFs are easier to implement and have fewer flaws and vulnerabilities;
- Security policy modelling pattern (ADV_SPMp) provides additional assurance from the development of a formal security policy model of the TSF and helps to gain correspondence between the functional specification and this security policy model;
- Preparative procedures pattern (AGD_PREp) presents how the TOE has been received and installed in a secure manner as intended by the developer;
- Operational user guidance pattern (AGD_OPEp) shows how to prepare material intended for all types of users of the TOE in its evaluated configuration;
- Functional tests pattern (ATE_FUNp) enforces the right specification, execution and documentation of tests;

- Test Coverage pattern (ATE_COVp) helps to demonstrate that the above mentioned TSFIs are properly covered by tests;
- Test Depth pattern (ATE_DPTp) helps to demonstrate that specified TOE design elements (subsystems, modules) are properly covered by tests;
- Independent testing pattern (ATE_INDp) has auxiliary meaning because the ATE_IND evidence is elaborated by evaluators; this pattern is used to verify the developer testing and performing additional tests by the evaluator.

The fourth group of evaluation evidences patterns is related to the development environment and the IT product or system developed there. This group of patterns concerns the AVA (Vulnerability assessment) assurance class which has only one family (AVA_VAN). This pattern is called Vulnerability analysis (AVA_VANp) and has auxiliary meaning as this kind of evidences (similarly to ATE_IND) are elaborated by evaluators. Please note that some vulnerabilities are TOE-inherent while some can be transferred from the development environment to the TOE.

The fifth group of evaluation evidences patterns is related to composition and their elaboration has been suspended at the moment.

3.2 Patterns Shape and Contents

Currently all patterns are defined as electronic documents editable with Microsoft Word. The patterns usually have a dozen or even more pages with well defined structures of data fields filled in by developers according to the precise guidance.

The patterns have their structures and contents elaborated on the basis of Common Criteria [14] Part 3 and the BSI Guidance [17]. The evidences related to the site-certification concepts were elaborated with the use of the [24] guide. All evaluation evidences were additionally verified against work units of the CC Evaluation methodology [16].

The hierarchical data structure of the given pattern includes different types of data fields placed within the CCMODE database. After the patterns validation, more advanced automation will be considered.

3.3 Patterns Validation and Examples

The complex validation of patterns is planned in the task 4, still several validations of the selected patterns were performed, with the focus on the elaboration of:

- Site Security Target for EMAG Security Lab,
- Security Target for a methane detector for mines,
- Security Target for a coal weighing system,
- Security Target for a hearing screening software application,
- ALC_LCD.1 evidence for the methane detector for mines,
- ALC_CMC.2 and ALC_CMC.4 evidences for the methane detector for mines,
- ALC_CMS.1 evidence for the methane detector for mines,
- ADV_ARC.1 evidence for the methane detector for mines,

- ADV_TDS.1 and ADV_TDS.2 evidences for the methane detector for mines,
- ADV_TDS.1 for the hearing screening software application,
- ADV_FSP.2 evidence for the methane detector for mines.

The patterns can be shown by a few examples from these validations.

Example 1: Site Security Target elaboration on the SSTp pattern basis

Figure 1 presents the pattern of the Site Security Target. On the left the SST structure is shown that should be filled in by the SST developer. The right panel presents one of the key SST sections with Security Problem Definition (SPD) according to the AST_SPD family. This section begins with an introductory text about SPD that can be supplemented by the developer. Please note subsections related to the assets, subjects and threats. The author should place the right enhanced generics into tables and supplement tables with his/her own comments. The contents of the SPD definition is formulated according to the on-line displayed hints.

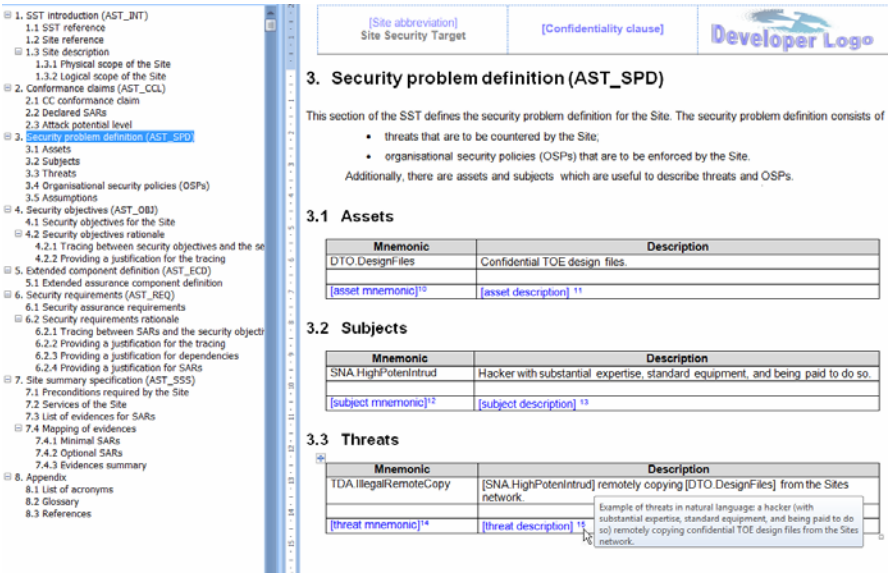


Fig. 1 Using the Site Security Target pattern (SSTp)

Please note a few enhanced generics placed in the figure, and a hint about threat description shown on the data field, marked by square brackets.

Example 2: Simple TOE design evidence

Figure 2 presents the ADV_TDSp pattern filled in with contents specific to the hearing screening software application.

It is the simplest case of this evidence due to the lowest rigour ADV_TDS.1 component used (EAL2). The left panel shows the structure of evidence, while the right one presents the TOE structure description. The developer placed an introductory text

and block scheme in the UML. Please note next fields that should be filled in with the proper data. Please note that the developer focuses only on describing the issues that are requested. He/she does not need to think about the structure and contents of evidences and the compliance with the CC standard.

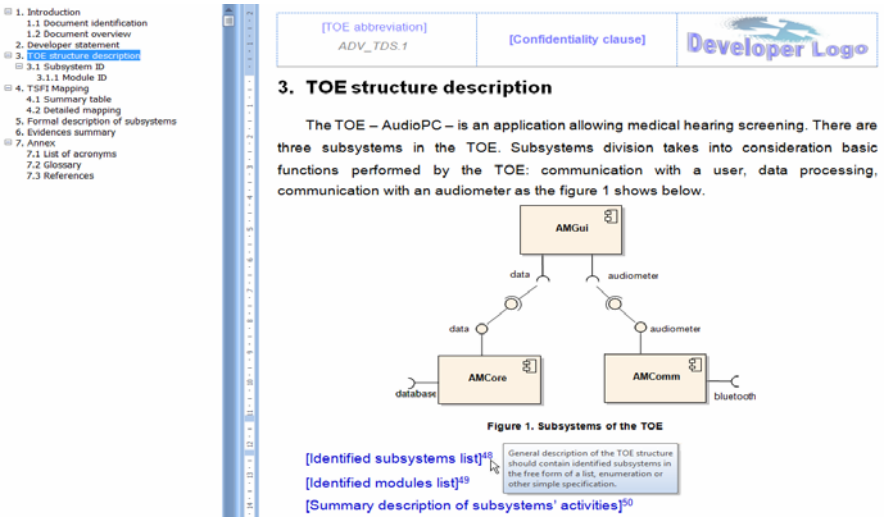


Fig. 2 Filling in the ADV_TDSp pattern for AudioPC application

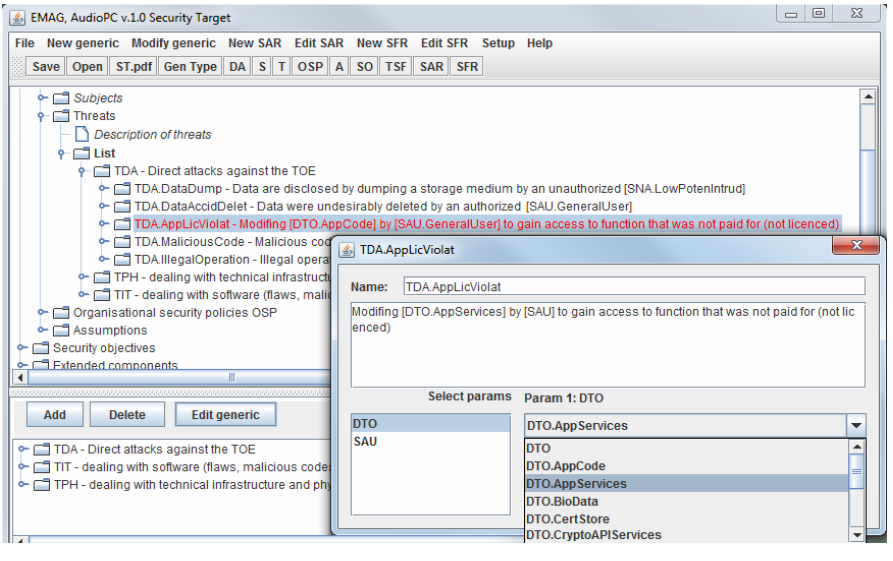


Fig. 3 Defining the security problem (SPD) for the AudioPC Security Target (ST)

Example 3: Specification means patterns

The example differs from the two above. It concerns patterns for the specification means, i.e. the author's defined enhanced generics and the standard defined components. Figure 3 presents a simple Java application for semi-automatic security target generation [20] in the form of pdf files. The main window has two panels. The upper one presents a design tree, i.e. its part presenting the Security Problem Definition – a few enhanced generics, specifying threats, are shown in Figure 3.

These generics are predefined as the specification means patterns and placed in the generics library (bottom panel). The developer selects the right generic and puts it into the design tree, this way defining the entire security target. The *TDA.AppLicViolat* generic is shown in the pop-up window. The *TDA* prefix means "direct attacks against the TOE" category. The string *AppLicViolat* is a mnemonic of the generic. The window contains the generic description with two parameters *DTO* and *SAU*. *DTO* means the TOE-related asset which was substituted by the *DTO.AppServices* generic selected from the list, while *SAU* means "authorized subject". The latter one was left uncompleted (means "any of SAU").

3.4 Patterns Customization

Currently the project works are focused on the customization processes of the evaluation evidences patterns. The customization of patterns dealing with the TOE development-, manufacturing- and maintenance processes embraces:

- identification of the developers' needs, restrictions, current standard incompliance and requirements,
- risk analysis within the development environment,
- transforming the Site Security Target pattern (SSTp) into the Site Security Target (SST) – for the "site-certification" option only,
- transforming patterns of evidences related to the TOE life-cycle support (mostly ALC-based) into the evidences,
- transforming the ISO/IEC 27001 patterns into the ISMS (Information Security Management System) documentation (an option),
- standard compliance and coherency checking.

The customization processes of the TOE development patterns encompass:

- transforming the Security Target pattern (STp) into the Security Target (ST) – similarly PP and their low assurance versions are processed,
- transforming patterns of evidences related to the TOE (ADV, AGD, ATE) into the evidences reflecting the character and technology of the elaborated IT product or system,
- coherency checking.

4 Conclusions

The chapter discusses briefly the IT security development process and presents the introduced patterns for evaluation evidences, along with the analyses provided to develop such patterns. The patterns usage is shown by a few examples being a part of a more complex case study made during the project. Please note that:

- the patterns have the right structure, verified against the CC standard and related documents,
- the contents of evidences is enforced by precise guidance (hints) elaborated on the basis of the CC standard and related documents;
- the developer fills in the pattern with knowledge related to the development environment or related to the developed IT product or system; he/she is asked about issues required exactly by evaluators,
- evidences are elaborated in a unified way and their presentation is also unified,
- some data fields of common meaning can be placed once, some fields describe specific issues for the given pattern; this facilitates change management and future automation.

The chapter concludes the presented pattern-based methodology with respect to the achieved and planned project results. Currently the elaborated basic set of patterns is validated. The implementation method with respect to the identified developers' needs and expectations has been elaborated as well.

CCMODE assumptions and achieved results were presented at the 11th International Common Criteria Conference [11]. Providing IT security developers with the CC-related security design patterns and related knowledge allows to:

- facilitate and speed up the IT security development process,
- improve the quality of evidences, because they are more consistent and include all details required by the considered assurance components and best practices,
- get computer support of the IT security development process.

CCMODE provides a new commercial solution, i.e. a methodology of building specialized environments for the development, production and maintenance of IT products and systems in compliance with rigorous regulations of ISO/IEC 15408 Common Criteria, called development environments. The chapter intents to encourage practitioners: developers, programmers and technology specialists responsible for IT security development to apply the project results in the future. It presents only an idea of the project products. The project results will be available on commercial basis according to EU regulations for this kind of projects.

References

[1] Bialas, A.: Common Criteria Related Security Design Patterns—Validation on the Intelligent Sensor Example Designed for Mine Environment. Sensors 10, 4456–4496 (2010)

[2] Bialas, A.: Intelligent Sensors Security. Sensors 10, 822–859 (2010)

[3] Bialas, A.: IT security development – computer-aided tool supporting design and evaluation. In: Kowalik, J., Górski, J., Sachenko, A. (eds.) Cyberspace Security and Defense: Research Issues, NATO Science Series II, vol. 196, pp. 3–23. Springer, Heidelberg (2005)

[4] Bialas, A.: Ontological approach to the IT security development process. In: Tkacz, E., Kapczynski, A. (eds.) Internet – Technical Development and Applications Series: Advances in Intelligent and Soft Computing, pp. 261–270. Springer, Heidelberg (2009)

[5] Bialas, A.: Ontological approach to the motion sensor security development. Electrical Review (Przeglkąd Elektrotechniczny) 85(R.85), 36–44 (2009)

[6] Bialas, A.: Security-related design patterns for intelligent sensors requiring measurable assurance. Electrical Review (Przeglkąd Elektrotechniczny) 85(R.85), 92–99 (2009)

[7] Bialas, A.: Ontology-Based Security Problem Definition and Solution for the Common Criteria Compliant Development Process. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T. (eds.) Proc. of Int. Conf. on Dependability of Computer Systems (DepCoS-RELCOMEX 2009, pp. 3–10. IEEE Computer Society, Washington (2009)

[8] Bialas, A.: Semiformal Approach to the IT Security Development. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T. (eds.) Proc. of the Int. Conf. on Dependability of Computer Systems (DepCoS-RELCOMEX 2007), pp. 3–11. IEEE Computer Society, Washington (2007)

[9] Bialas, A.: Semiformal Common Criteria Compliant IT Security Development Framework. In: Stud. Inf., vol. 292(2B(77)). Silesian University of Technology Press, Gliwice (2008), http://www.znsi.aei.polsl.pl/ (accessed on January 2, 2011)

[10] Bialas, A.: Semiformal Framework for ICT Security Development. In: Presentation on the 8th International Common Criteria Conference, Rome, Italy, September 25–27 (2007)

[11] Bialas, A.: Patterns-based development of IT security evaluation evidences. In: The 11th Int. Common Criteria Conf., Antalya, http://www.11iccc.org.tr/ (accessed 3 January 3, 2011)

[12] Bialas, A.: Validation of the Specification Means Ontology on the Simple Firewall Case. In: Proc. of the Int. Conf. on Security and Management, Las Vegas, vol. 1, pp. 278–284 (2009)

[13] CCMODE project home page, http://ccmode.emag.pl/ (accessed January 3, 2011)

[14] Common Criteria for IT Security Evaluation version 3.1, part 1-3 (2009), http://www.commoncriteriaportal.org/ (accessed January 3, 2011)

[15] Common Criteria Portal home page, http://www.commoncriteriaportal.org/ (accessed January 3, 2011)

[16] Common Evaluation Methodology for Information Technology Security version 3.1 (2009), http://www.commoncriteriaportal.org/ (accessed January 3, 2011)

[17] Guidelines for Developer Documentation according to Common Criteria version 3.1, Bundesamt für Sicherheit in der Informationstechnik (2007)

[18] Hermann, D.S.: Using the Common Criteria for IT Security Evaluation. CRC Press, Boca Raton (2003)

[19] Juerjens, J.: Secure Systems Development with UML. Springer, Heidelberg (2005)
[20] Nowak, P.: Oprogramowanie do wspomagania konstruowania zabezpieczeń teleinformatycznych wykonane zgodnie z metodyką Common Criteria w technologii Java/XML. Instytut Informatyki Politechniki Śląskiej, Gliwice (the master thesis) (2009)
[21] Noy, N F., McGuiness, D L.: Ontology Development 101: A Guide to Creating Your First Ontology, Knowledge Systems Laboratory (2011), http://www-ksl.stanford.edu/people/dl/papers/ontology-tutorial-noy-mcguinness-abstract.html (accessed January 2, 2011)
[22] Protégé Ontology Editor and Knowledge Acquisition System, Stanford University, http://protege.stanford.edu/ (accessed January 2, 2011)
[23] Schumacher, M., Fernancez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerland, P.: Security Patterns: Integrating Security and Systems Engineering. John Wiley and Sons, Chichester (2006)
[24] Site Certification. Supporting Document Guidance (2007) version 1.0, revision 1 (CCDB-2007-11-001), http://www.commoncriteriaportal.org/ (accessed January 2, 2011)
[25] Yoshioka, N., Washikazi, H., Maruyama, K.: A survey on security patterns. Progress in Informatics 5, 35–47 (2008)

A Comparison of Dataflow and Mutation Testing of Java Methods

Ilona Bluemke and Karol Kulesza

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: I.Bluemke@ii.pw.edu.pl

Abstract. The objective of this chapter is to compare the dataflow and the mutation testing of several Java programs. Experiments were conducted in the Eclipse environment. DFC plugin was used to support the dataflow testing while MuClipse and Jumble plugins were used for the mutation testing. The results of testing six Java programs using data flow and mutation techniques are presented. Experiment shown, that the effectiveness of mutation testing is higher than the effectiveness of dataflow testing. Mutation technique appeared also to be more expensive than the data flow one, if time and effort are considered.

1 Introduction

Effective testing is one of the key issues in developing dependable software systems. Popular approaches to testing include "black box" and "white box". In white box approach the test cases can be derived from the code of the unit under test. Code based approach can be divided into two main types: data flow coverage methods [9, 10, 26, 28] and control flow coverage e.g. [18, 30]. In the dataflow testing relationships between data are used to select test cases. Such approach was introduced for structural programming languages by Rapps and Weyuker [26] and later adopted for object languages [9, 10, 28,]. Dataflow testing of Java programs is also described in [2, 24].

Mutation testing is a fault based software testing technique that was introduced more than thirty years ago. The general idea is that the faults used in mutation testing represent the mistakes made by a programmer so they are deliberately introduced into the program to create a set of faulty programs called *mutants*. Each mutant program is obtained by applying a mutant operator to a location in the original program. To assess the quality of a given set of tests these mutants are executed against the set of input data to see, if the inserted faults can be detected. A very good survey of mutation techniques was written in 1996 by Jia and Harman [12].

Since their introduction, mutation and data flow testing were considered as potentially effective but there were very few experiments examining the effectiveness of these methods. Several researchers conducted empirical studies in years

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 17–30.
springerlink.com

1993-1996 on Fortran, Pascal and C programs. They evaluated various measures, including the effectiveness at exposing faults, the difficulty and the cost of usage. The results of these experiments are presented in section 3. The moment we started our experiment there were no similar experiments conducted on Java programs. We wanted to check, if the results obtained for Fortran and C programs are still valid for Java programs.

The objective of this chapter is to compare the dataflow and the mutation testing of several Java programs. The main ideas of dataflow and mutation testing are briefly described in section 2 while related work is presented in section 3. Our experiments were conducted in the Eclipse environment. DFC (Data Flow Cover) plugin implemented at the Institute of Computer Science, Warsaw University of Technology [2, 25], was used to support the dataflow testing. MuClipse [21] and Jumble[13] plugins were used for the mutation testing. The results of experiments are presented in section 4 and some conclusions are given in section 5.

2 Data Flow and Mutation Testing

Below the basics information on the data flow and the mutation testing are given.

2.1 Data Flow Testing

The dataflow testing is one of "white box" techniques, the test cases are derived from the source code. In the dataflow testing [9, 26] the relations between data are the basis to design test cases. Different sub-paths from *definition* of a variable (e.g. assignment) into its *use* are tested. A *definition-use* pair (denoted as *def-u*) is an ordered pair <*s, u*>, where *s* is a statement containing the definition of the variable *v,* and *u* is a statement containing the use of *v*, or some memory location bound to *v*, that can be reached from *s* over some program path. For code given in appendix on listing 1, *def-u* pairs for variable customer are following: <30,44>, <30,55>, <30,53>, <30,34>, <30,45>.

The test criteria are used to select particular *def-u* pairs. A test satisfies *def-u* pair, if executing the program with this test causes the traversal a sub-path from the definition to the use of this variable *v* without any *v* redefinition. A *def-u* pair is *feasible* if exists some program input that will cause it to be executed. The dataflow testing criteria [26] use the *def-use graph* (denoted as DUG), which is derived from the control flow graph. On the control flow graph the information about the set of variables defined - *def()* and used - *use()* in each node/edge are added. Many *def-u* criteria have been proposed and compared [1, 2, 4]. One criterion, called *all-defs* states, that for each DUG node *i* and all variables *v*, defined in this node, $v \in def(i)$ at least one path <*i, j*> is covered. In node *j* this variable is used $v \in use(j)$ and on this path the variable *v* is not redefined.

The first dataflow technique [26] was proposed to structural programming languages and does not consider dataflow interactions that arise when methods are invoked in an arbitrary order. In [10] an algorithm was proposed, called PLR, to find *def-u* pairs, if the variable definition is introduced in one procedure, and the variable usage is in called or calling procedures. The algorithm works on inter-procedural control flow graph built from control flow graphs of dependent procedures. This method can be adapted to global variables, class attributes and referenced method arguments in testing object programs.

For object programs three levels of dataflow testing were proposed in [9]: *intra-method, inter-method, intra-class.* For each of these testing levels appropriate *def-u* pairs were defined i.e. *intra-method*, *inter-method* and *intra-class*.

The data flow testing would be a tedious work without a proper tool support. Among many tools supporting code based testing of object programs, only JaBUTi [11] and ***DFC*** (Data Flow Coverage) [2] are dedicated to Java. ***DFC*** was implemented as an Eclipse plugin at the Institute of Computer Science, Warsaw University of Technology. DFC [2] finds all definition-uses pairs in tested unit and provides also the definition-uses graph for methods. After the execution of each test, the tester is provided with the information which *def-u* pairs were covered so can add new tests for not covered pairs. The tester can also configure the DFC, decide which methods are changing the state of an object.

2.2 Mutation Testing

The mutation testing is a fault based software testing technique that was introduced in 1971 by Richard Lipton (according to [20]). A very good survey of mutation techniques was written in 1996 by Jia and Harman [12].They also constructed mutation testing repository [22] with many papers on mutation testing.

The general idea of mutation testing is that the faults used represent mistakes made by a programmer, so they are deliberately introduced into the program to create a set of faulty programs called *mutants*. Each mutant program is obtained by applying a mutant operator to a location in the original program. Typical mutation operators include replacing one operator e. g. '+' by another e.g. '-' or replacing one variable by another. To assess the quality of a given set of tests the mutants are executed on a set of input data to see, if the inserted faults can be detected. If the test is able to detect the change (i.e. one of the tests fails), then the mutant is said to be *killed*. The input data for test should cause different program states for the mutant and the original program.

A variety of mutation operators were explored by researchers. Below are given some examples of mutation operators for imperative languages:

- statement deletion,
- replacement of each boolean subexpression with *true* and *fals*,
- replacement of each arithmetic operation with another one, e.g.: "*" with " /",
- replacement of each boolean relation with another one, e.g.: > with >=, == .

These mutation operators are also called traditional mutation operators. There are also mutation operators for object-oriented languages e.g. [3, 8, 24], for concurrent constructions, complex objects like containers etc., they are called class-level mutation operators. In appendix in listing 2 the mutated line, for code given in listing 1, is shown.

Many mutation operators can produce *equivalent mutants*. The resulting program is equivalent to the original one and produces the same output as the original one. Determining which mutants are equivalent is a tedious activity, usually not implemented in tools. An example of equivalent mutant is given in appendix in listing 3, for code shown in listing 1.

Mutation score is a kind of quantitative test quality measurement that examines a test set's effectiveness. It is defined as a ratio of the number of killed mutants to the total number of non-equivalent mutants (1).

$$mutation\ score(P,T) = \frac{K}{M - E} \tag{1}$$

Where: P – tested program, T – set of tests, K – number of killed mutants, M – total number of mutants, E – number of equivalent mutants.

The total number of nonequivalent mutants results from the difference between the total number of mutants and the number of equivalent mutants which cannot be killed.

Mutation testing can be used for software testing at the unit [1, 3, 6, 7] or integration level [5]. It has been applied to many programming languages, as a white box unit testing: for example Fortran programs [7], C# [6], C [1], Java [3]. Mutation testing of software would be difficult without a reliable, fast and automated tool that generates mutants, runs them against a test suit and reports the results. Among several Java mutation tools we choose Muclipse[21] and Jumble [13], Eclipse plugins.

3 Related Work

To our best knowledge the first empirical comparison of data flow and mutation testing was preformed by Mathur and Wong [19]. They randomly generated 30 adequate test sets for each of three variants of mutation analysis and for *all-uses* for each of five programs. They did not report on the size of the test sets but it can be expected that the mutations sets are much larger than the *all-uses* ones. In all ten of their subjects mutation testing was at least as effective as *all-uses*. Less expensive variants of mutation testing were reported by them as superior to *all-uses*. They did not use any statistical inference procedure so it is not clear whether the reported advantage is statistically significant. The high effectiveness they demonstrated for mutation testing may be an artifact of large test set sizes.

In [23] Offutt et al. described an experiment in which they evaluate the data flow and the mutation testing. They chose 10 Fortran program that cover a range of applications e.g. bubble sort, transitive closure of a matrix, median of three integers). These programs were rather small (size from 10 to 29 executable

statements), have from 183 to 3010 mutants, and have from 10 to 101 *def-u* pairs. They also give the number of *def-u* pairs and the number of infeasible *def-u* pairs, the number of mutants and equivalent mutants. Because of the nature of the two testing techniques, programs typically have many more mutants than *def-u* pairs. They observed, that the test cases overlap: in the sense that one test case usually kills many mutants, and often covers several *def-u* pairs. Mutation tool for Fortran – Mothra, and ATAC - a data flow tool for C were used in the experiment, so Fortran programs have to be translated into C language. The requirements for mutation test cases was killing mutants while executing *def-u* pairs were test requirements for the data flow testing. Both mutation and data flow test have problems with unrealizable test requirements. Mutation systems create equivalent mutants, which cannot be killed, and data flow systems ask for infeasible *def-u* pairs to be covered. For each program, as a part of preparation, they identified all equivalent mutants and infeasible *def-u* pairs "by hand".

They presented results from three empirical studies. Firstly, they compared mutation with *all-uses* data flow on the basis of a "cross scoring", where tests generated for each criterion were measured against the other one. Secondly, they measured the fault detection of the test data generated for each criterion, and compared the results. Thirdly, they compared the two testing techniques on the basis of the number of test cases generated to satisfy them, in a rough attempt to compare their relative costs. For these programs, the mutation scores for the data flow-adequate test sets were reasonably high, with an average coverage of mutation by data flow of 88.66%. While this implies that a program tested with the *all-uses* data flow criterion has been tested to a level close to mutation-adequate, it may still have to be tested further to obtain the testing strength afforded by mutation.

However, the mutation-adequate test sets come very close to covering the data flow criterion. The average coverage of data flow by mutation was 98.99% for their ten programs. They infer, that a program that has been completely tested with mutation analysis will usually be very close to having been tested to the *all-uses* data flow criterion, within one or two *def-u* pairs of being complete. On the other hand, mutation required more test cases in almost every case than the data flow testing did, providing a cost to be the trade of these two techniques. These conclusions are supported by the faults that the test sets detected. Although both mutation adequate and data flow-adequate test sets detected significant percentages of the faults, the mutation-adequate test sets detected an average of 16% more faults than the data flow-adequate test sets. The difference was as high as 60% for one program. This provides some evidence that mutation is better than *all-uses* data flow. These results are in general in an agreement with those of Mathur and Wong [19]. Both studies found that mutation offers more coverage than data flow, but at a higher cost. Although Mathur' and Wong' study did not include any fault detection, Offutt et al. also found, that mutation-adequate test sets detected more faults. The fact that both studies, performed at about the same time by different researchers using different programs and test cases, got similar results, greatly strengthens these conclusions.

Other experiments were conducted by Frankl, Weiss and Hu in 1996 [7]. They were testing ten programs (obtained from seven) rather simple (the LOC for these

programs were in the range 22-78). Pascal versions were used for the *all-uses* subject and equivalent Fortran versions, as close as possible in structure, were used for mutation testing.

In [16] *all-uses* and mutation testing of Java classes was also compared. The authors were testing twenty-nine Java small classes. They were taken from various sources, including open source software websites, the accompanying CD to the Java or testing textbook. Faults were seeded into the Java classes by one of the authors. The testing process was as follows. The tests to satisfy *all-uses*, were ran on the mutants, and tests that did not contribute to mutation coverage were eliminated. Finally, additional tests to kill the remaining mutants were added. The results of this experiment indicate, that mutation testing found more faults than other criteria. A little surprising to the authors was that, despite its widespread reputation as being the "most expensive test criterion," mutation did not require significantly more tests. In [16] authors wrote, that the expense of mutation is worthwhile, because it will help the tester to find more faults. The author claim also that the faults that mutation did not catch, can be instructive; it may be possible to design additional mutation operators that can detect these faults.

Other interesting experiment on mutation and coverage testing of Java programs was conducted in 2009 and is presented by Madeyski in [17]. The goal of this experiment was different, than those described above. In this experiment the impact of the "test first" programming on mutation score was examined. Additionally, code coverage was measured to get a more complete picture of the experimental manipulation. Branch coverage and mutation score indicator were examined to get some insights how thoroughly unit tests exercise programs, and how effective they are. The main result of this experiment is that the "test first" programming practice, used instead of the classic "test last" technique, does not significantly affect the branch coverage and the mutation score indicator.

An overview of unit testing experiments is given in [15].

4 Comparison of Dataflow and Mutation Testing

The goal of our experiment was to compare the mutation and the dataflow testing of Java code. When we started our study there were no similar experiments reported in the literature. Following tools were used in the experiment: DFC (Data Flow Cover) [2, 25] for data flow testing, for mutation testing MuClipse [21] and Jumble[13] plugins and JUnit [14].

Six Java classes were used as experiments subjects. They were taken from various sources, including websites and author's private repositories. Since in DFC only intra-method data flow testing is performed we used only six methods. We choose simple methods but with nontrivial control flow graph and for which the def-use graph contained several branches. The methods used in experiment are listed in Table1. The methods used in other similar experiments (section 3) were of comparable complexity.

MuClipse and Jumble generate mutants, exercise them on a test cases prepared by the tester, and after a series of executed test show, which mutants are alive and which one are killed. The mutation score for a class is also calculated. These tools do not calculate the mutation score for a method. To calculate the mutation score

Table 1 Tested methods

	Class.method	*project*	*source*	*LOC*
1	CoffeeMaker.addRecipe	CoffeeMaker	[4]	23
2	Shop.doShopping	Shop	[5], appendix listing 1	26
3	Bank.grantCredit	theBank	private	20
4	Board.insertShip	NetworkShipBattl	[27]	43
5	PizzaClub.makePizza	thePizzaClub	private	20
6	Library.borrowBook	theLibrary	private	21

for a method the tester has to identify mutants for this method, and next calculate the ratio of the number of killed mutants to the total number of non-equivalent mutants ("by hand"). Unfortunately MuClipse and Jumble are not able to find equivalent mutants automatically, so the code analysis of mutants by the tester was necessary. As this is a tedious activity, especially in MuClipse with so many generated mutants only six simple methods were the subject of this experiment.

The DFC [2, 25] Eclipse plugin instruments the source Java code (adds extra instructions needed for finding dataflow coverage) and builds def-use graph (DUG). DUG contains information concerning the control flow, variable definitions and usage in its nodes. Instrumented code should be compiled and run in Eclipse environment on a set of test cases prepared by the tester. The extra code added by a module of DFC sends the data concerning the coverage to DFC. Another module of DFC is locating covered and not covered *def-u* pairs. After the execution of a test, the tester is provided with the information which *def-u* pairs were covered so can add new tests for not covered ones. The tester decides which test criteria to use e.g. *all-defs*. In our experiment the *all-uses* criterion was used. The *all-uses* criterion requires tests to tour at least one subpath from each definition to each reachable use. In *DFC* in the *"standard"* configuration, all methods are initially identified as not modifying the state of the object and using it. In *"optional"* configuration the tester manually identifies defining and using methods. Details on DFC implementation and its usage can be found in [25]. The *"infeasible"* pairs, not possible to cover by test, were identified by the tester *"by hand"*, but with the information provided by DFC (e.g. DUG graph), we found it easier and faster than finding equivalent mutants.

Test cases were prepared separately for the mutation and the data flow testing. The goal of testing was to kill all nonequivalent mutants and to cover all feasible *def-u* pairs. Different approach was used in [16], where the tests satisfying *all-uses*, were ran on the mutants, tests not contributing to mutation coverage were eliminated and additional tests to kill the remaining mutants were added.

In Table 2 the number of mutants obtained for each tested method by MuClipse and Jumble plugins are given. The number of mutants generated by MuClipse is greater than the number of mutants generated by Jumble, because the number of mutation operators available in MuClipse is 43 (15 at the class level and 28 at the method level), is also significantly greater than in Jumble (only 7).

The effectiveness of mutation testing is usually measured as mutation score (section 2, equation (1)). We introduce similar formula for the data flow testing:

$$data\ flow\ score(P,T) = \frac{C}{PDU - I} \tag{2}$$

Where: P – tested program, T – set of tests, C – number of covered *def-u* pairs, PDU – total number of *def-u* pairs, I – number of infeasible *def-u* pairs.

Table 2 Mutants and def-u pairs for methods

Number of Class.method (Table 1)	mutants		Equivalent mutants		def-u pairs		infeasible def-u pairs	
	MuClipse	Jumble	MuClipse	Jumble	conf.	conf. optional	conf.	conf. optional
1	44	20	2	1	32	32	9	9
2	58	13	4	0	31	38	2	3
3	108	24	8	0	29	28	2	0
4	218	52	8	3	65	65	4	4
5	94	22	12	0	34	35	5	5
6	47	15	2	0	34	27	11	3

We conducted several experiments comparing different aspects of data flow and mutation testing. One of such experiments comparing the mutation testing and the data flow testing using mutation score (1) and data flow score (2) is described in next section.

4.1 Experiment

In the experiment we wanted to check what efficiency in mutation testing can be obtained using the test covering all feasible *def-u* pairs and vice versa. Tests, covering all feasible *def-u* pairs, were given as input to mutants generated by MuClipse and Jumble.

Let's T_M denote the set of test cases killing all non equivalent mutants and T_{DF} the set of test cases covering all feasible *def-u* pairs. The goal of this experiment can be expressed as comparing the *mutation score (P, T_{DF})* and *data flow score (P, T_M)*.

The *mutation score (P, T_{DF})* for mutants generated by MuClipse and Jumble and for both possible DFC configurations are given in Table 3. The average values for all tested methods are 87,84% for MuClipse and 93,62% for Jumble. These values do not depend on the DFC configuration. In the "optional" DFC configuration (the tester marks methods as defining or using object state [2, 25]) sometimes new test cases are necessary to cover the specific data flow. These test cases in methods from Table 1 were not able to 'kill' additional mutants. In "killing" mutants often boundary data values are successful and such values not appeared in the added test cases.

Table 3 Mutation score for mutants generated by MuClipse and Jumble on test sets designed for DFC

Class.method	*Mutation score obtained for test sets designed for DFC in standard configuration*		*Mutation score obtained for test sets designed for DFC in optional configuration*	
	MuClipse	*Jumble*	*MuClipse*	*Jumble*
CoffeeMaker.addRecipe	95,24%	100%	95,24%	100%
Shop.doShopping	85,19%	100%	85,19%	100%
Bank.grantCredit	74%	79,16%	74%	79,16%
Board.insertShip	87,61%	95,91%	87,61%	95,91%
PizzaClub.makePizza	93,90%	100%	93,90%	100%
Library.borrowBook	91,11%	86,67%	91,11%	86,67%
	87,84%	**93,62%**	**87,84%**	**93,62%**

Table 4 Data flow score and mutation score for test cases designed for MuClipse

Class.method	*Mutation score for Jumble*	*Data flow score*	
		Standard conf.	*Optional conf.*
CoffeeMaker.addRecipe	100%	100%	100%
Shop.doShopping	100%	100%	85,71%
Bank.grantCredit	91,67%	96,30%	96,43%
Board.insertShip	89,80%	100%	100%
PizzaClub.makePizza	100%	93,10%	90%
Library.borrowBook	93,33%	91,30%	91,67%
	95,80%	**96,78%**	**93,97%**

Table 5 Data flow score and mutation score for test cases designed for Jumble

Class.method	Mutation score for MuClipse	Data flow score	
		Standard conf.	Optional conf.
CoffeeMaker.addRecipe	95,24%	100%	100%
Shop.doShopping	83,33%	93,10%	74,29%
Bank.grantCredit	72%	96,30%	96,30%
Board.insertShip	96,19%	100%	100%
PizzaClub.makePizza	95,12%	93,10%	90%
Library.borrowBook	86,76%	78,26%	79,17%
	88,11	**93,46%**	**89,96%**

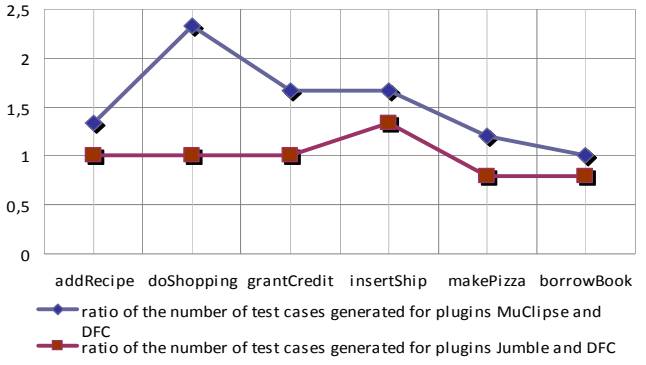


Fig. 1 Ratio of the number of test cases in mutation tools and DFC in standard configuration

In Table 4 and Table 5 the *data flow score (P, T_M)* for test sets designed for MuClipse and Jumble are given. The average value of *data flow score (P, T_M)* for test sets from MuClipse is very high - 96,78%. For three methods the test cases covered all feasible *def-u* pairs. Such high value can be obtained because the number of MuClipse mutation test cases is significantly higher than the data flow test cases (Fig. 1).

For Jumble plugin the number of test cases is smaller (Fig. 1) and close to the number of test cases covering *def-u* pairs, so lower is also the *data flow score (P, T_M)* i.e. 93,46% for standard and 89,96% for optional configuration. The lower values of *data flow score(P, T_M)* for "optional" DFC configuration are caused by new *def-u* pairs which often appear in this configuration. and T_M. tests were not able to cover them. For the method `doShopping` (appendix - listing 1) the difference of values of the *data flow score* is significant (from 93% into 74%).

4.2 Results of Experiments

The results of experiment show that *data flow score(P, T_M)* depends on the mutation tool used. Test prepared for MuClipse are able to provide high values of *data flow score(P, T_M)*, higher than *mutation score (P, T_{DF})* – Fig. 2.

Tests prepared for Jumble, providing only seven mutation operators, are not able to cover all *def-u* pairs, especially when the tester marks methods as defining or using object state (optional DFC configuration). Such activity causes usually the increase in the number of *def-u* pairs. In Fig. 3 we can see that only for two methods (`grantCredit, insertShip`) *data flow score(P, T_M)* is higher than *mutation score (P, T_{DF})* .

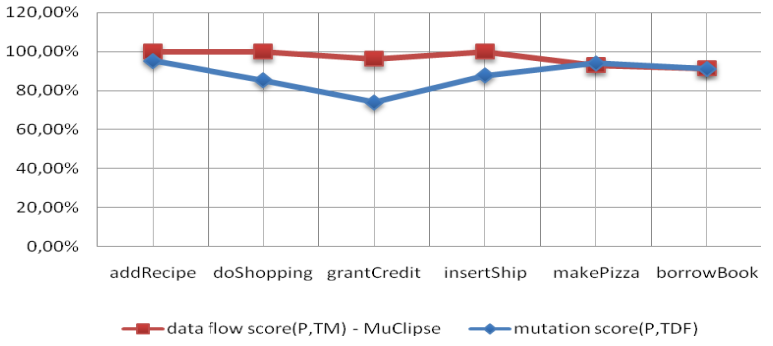


Fig. 2 *data flow score* (P,T_M) and *mutation score* (P,T_{DF}) – MuClipse and standard DFC configuration

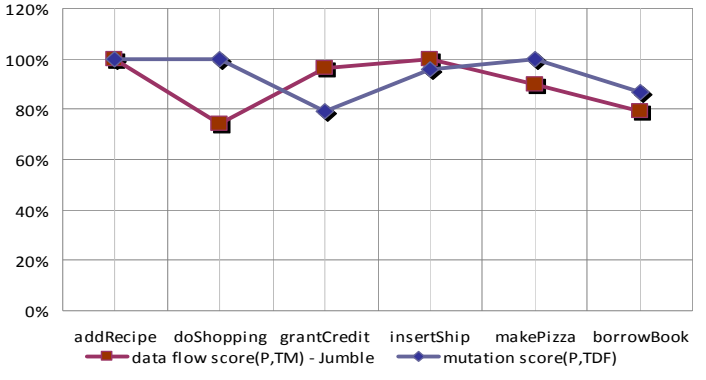


Fig. 3 *data flow score* (P,T_M) and *mutation score* (P,T_{DF}) – Jumble and optional DFC configuration.

In Table 4 and Table 5 mutation scores for mutants generated by one mutation tool and tested with test cases designed for other tool are also given. Tests designed for MuClipse were able to kill in average 95,80% Jumble' mutants, for three out of six methods 100% mutants were killed.

The results obtained by testing MuClipse mutants by test prepared for Jumble mutants, are not as good. Only 88,11% of mutants created by MuClipse were killed by tests designed for Jumble mutants. As MuClipse uses many mutation operators the test cases designed to kill them are very effective in killing mutants generated by tool with fewer number of mutation operators and in covering def-u pairs.

5 Conclusions

In this chapter we compared *all-uses* data flow testing with mutation testing on six simple Java applications. To our best knowledge where our research started there

were no similar experiments described in the literature. For the simple methods used in our experiments, the data flow scores for mutation adequate test sets are reasonably high, higher than 90% for mutation tool with many mutation operators. Mutation scores for data flow adequate test sets are slightly lower (about 80%) but also acceptable in many applications. In 2009 the results of similar experiment (described in section 3) which are very close to ours, were published.

Our study has several limitations. As in all studies that use software as subjects, external validity is limited by the number of subjects and the fact that we have no way of knowing whether they are representative of the general population. All the classes were quite simple, as in other experiments (section 3), and we must leave it to a future replicated study to see if the results would be similar for larger and more complicated classes.

If results like ours, those of Mathur and Wong and of Offut et al. could be show to scale up to larger programs and to be statistically significant, they would provide an argument in favor of choosing mutation testing over all-uses, if cost were not a factor. We estimate that the effort needed for mutation testing with MuClipse is twice, or even more bigger than the effort in data flow testing.

Acknowledgments. We are very grateful to the reviewers for many valuable remarks.

References

[1] Agraval, H., Demilo, RA., et. al: Design of Mutant Operators for the C Programming Language. Purdue University, West Lafayette, Technical Report SERC-TR-41-P (1989)

[2] Bluemke, I., Rembiszewski, A.: Dataflow approach to testing Java programs. In: Proceedings of International Conference on Dependability of Computer Systems DepCoS - RELCOMEX 2009, pp. 69-76 (2009), ISBN 978-0-7695-3674-3

[3] Chevalley, P., Thevenod-Fosse, P.: Mutation Analysis Tool for Java Programs. International Journal on Software Tools for Technology Transfer 5(1), 90–103 (2002)

[4] Coffeemaker, `http://agile.csc.ncsu.edu/SEMaterials/tutorials/coffee_maker` (accessed 2010)

[5] Delamaro, M.E., Maldonado, J.C.: Interface Mutation: Assessing Testing Quality at Interprocedural Level. In: Proceedings of the International Conference of the Chilean Computer Science Society (SCCC 1999), Talca, Chile, pp. 78–86 (1999)

[6] Derezińska, A., Szustek, A.: Tool supported Advanced Mutation Approach for Verification of C# Programs. In: Proceedings of International Conference on Dependability of Computer Systems DepCoS - RELCOMEX 2008, Szklarska Poręba, Poland, pp. 261–268 (2008)

[7] Frankl, P.G., Weiss, S.N., Hu, C.: All-uses versus mutation testing: An experimental comparison of effectiveness. Journal of Systems and Software 38, 235–253 (1997)

[8] Garhwal, S., Kumar, A., Sehrawat, P.: Mutation Testing for Java. In: Proceedings of National Conference on Challenges & Opportunities in Information Technology (COIT-2007), pp. 22–28 (2007)

[9] Harrold, M.J., Rothermel, G.: Performing data flow testing on classes. In: Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 154–163 (1994)

[10] Harold, M.J., Soffa, M.L.: Interprocedural data flow testing. In: Proceedings of the Third Testing, Analysis, and Verification Symposium, pp. 158–167 (1989)
[11] Jabuti, http://jabuti.incubadora.fapesp.br/ (accessed 2009)
[12] Jia, Y., Harman, M.: An Analysis and Survey of the Development of Mutation Testing. Technical report TR-09-06, Crest Centre, Kong's College London (1996), http://www.dcs.kcl.ac.uk/pg/jiayue/repository/TR-09-06.pdf (accessed 2010)
[13] Jumble, http://jumble.sourceforge.net/index.ht (accessed 2010)
[14] Junit, http://www.junit.org/ (accessed 2010)
[15] Juristo, N., Moreno, A.M., Vegas, S., Solari, M.: Search of What We Experimentally Know about Unit Testing. IEEE Software, 72–80 (2006)
[16] Li, N., Praphamontripong, U., Offutt, J.: An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-uses and Prime Path Coverage. In: Proceedings of the 4th International Workshop on Mutation Analysis (MUTATION 2009), Denver, Colorado (2009)
[17] Madeyski, L.: The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment. Information and Software Technology 52(2), 169–184 (2010)
[18] Malevris, N., Yates, D.F.: The collateral coverage of data flow criteria when branch test-ing. Information and Software Technology 48, 676–686 (2006)
[19] Mathur, A.P., Wong, W.E.: An empirical comparison of data flow and mutation-based test adequacy criteria. The Journal of Software Testing, Verification, and Reliability 4(1), 9–31 (1994)
[20] Mathur, A.P.: Mutation Testing. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering, pp. 707–713 (1994)
[21] Muclipse, http://muclipse.sourceforge.net/index.php (accessed January 2011)
[22] Mutation repository, http://www.dcs.kcl.ac.uk/pg/jiayue/repository (accessed January 2011)
[23] Offutt, A.J., Pan, J., Tewary, K., Zhang, T.: An Experimental Evaluation of Data Flow and mutation Testing. Software Practice and Experience 26(2), 165–176 (1996)
[24] Offutt, A.J., Untch, R.H.: Mutation 2000: Uniting the Orthogonal. In: Wong, W.E. (ed.) Mutation testing for the new century, pp. 34–44. Kluver Academic Publishers, Boston (2001)
[25] Rembiszewski, A.: Testing object programs by data flow coverage. MSc thesis, Institute of Computer Science, Warsaw University of Technology (2009)
[26] Rapps, S., Weyuker, E.J.: Selecting software test data using data flow information. IEEE Transactions on Software Engineering 11, 367–375 (1985)
[27] Suchowski, J.: Network game – NetworkShips Battle. Institute of Computer Science, Warsaw University of Technology (2010)
[28] Vincenzi, A.M.R., Maldonado, J.C., Wong, W.E., Delamaro, M.E.: Coverage testing of Java programs and components. Science of Computer Programming 56, 211–230 (2005)
[29] Vincenzi, A., Delamaro, M., Höhn, E., Maldonado, J.C.: Functional, control and data flow, and mutation testing: Theory and practice. In: Borba, P., Cavalcanti, A., Sampaio, A., Woodcook, J. (eds.) PSSE 2007. LNCS, vol. 6153, pp. 18–58. Springer, Heidelberg (2010)
[30] Woodward, M.R., Hennell, M.A.: On the relationship between two control-flow coverage criteria: all JJ-paths and MCDC. Information & Software Technology 48, 433–440 (2006)

Appendix

```
30)  public Bill doShopping( theShop.Customer customer ) {
31)      Bill bill = new theShop.Bill();
32)      for (int i = 0; i < items.length; ++i) {
33)          Item item = items[i];
34)          if (customer.need( item )) {
35)              bill.add( item.getPrice() );
36)          }
37)      }
38)
39)      double vatAmount = vatPercent / 100 * bill.getTotalSum();
40)      if (addVat) {
41)          bill.add( vatAmount );
42)      }
43)
44)      double discountAmount = bill.getTotalSum() *
                     (customer.getDiscountPercent() / 100);
45)      if (customer.isSpecial()) {
46)          if (bill.getTotalSum() > minSumForDiscount) {
47)              bill.subtract( discountAmount );
48)          }
49)      }
50)
51)      bill.close();
52)
53)      if (bill.getTotalSum() <= customer.getMoneyAmount()) {
54)          bill.pay();
55)          customer.getFromAcount( bill.getTotalSum() );
56)      } else {
57)          bill.cancel();
58)      }
59)      return bill;
60)  }
```

Listing 1. Source code of method `doShopping`

```
53)  if (bill.getTotalSum() <= customer.getDiscountPercent()) {
```

Listing 2. Mutant EAM (accessor method change) getMoneyAmount() -> getDiscountPercent()

```
41)              bill.add( vatAmount++ );
```

Listing 3. Equivalent mutant - postincrement operator

A New Three Levels Context Based Approach for Web Search Engines Evaluation

Abdelkrim Bouramoul[1], Mohamed-Khireddine Kholladi[1], and Bich-Lien Doan[2]

[1] Computer Science Department, Misc Laboratory, University of Mentouri Constantine, B.P. 325, Constantine 25017, Algeria
e-mail: a.bouramoul@yahoo.fr, kholladi@yahoo.fr
[2] Computer Science Department, SUPELEC, Rue Joliot-Curie, 91192 Gif Sur Yvette, France
e-mail: bich-lien.doan@supelec.fr

Abstract. Classical approaches for evaluating information retrieval tools have limitations and shortcomings, particularly regarding to the consideration of the user, the manner in which these approaches measure the adequacy between the user's query and the returned documents, and the consideration of the search tool characteristics. This critical finding prompted our reflections for the exploitation of contextual elements around the user, the query and the search tool during the evaluation process. This paper presents a new approach based on three complementary levels of context for evaluating information retrieval tools. The experiments gives at the end of this article has shown the applicability of the proposed approach to real research tools. The tests were performed with the most popular searching engine (i.e. Google, Bing and Yahoo) selected in particular for their high selectivity. The results have revealed that the evaluation of search engines at three different levels of context is a promising way to diagnose the performance, characteristics and behavior of these engines and the relevance of the results that they return.

1 Introduction

Information Retrieval Systems (IRS) are tools for finding information in closed document collections or on the Web. The challenge is to finding, among the large volume of documents available, those that best fit our needs. Consequently several questions arise about these information retrieval tools, particularly in terms of their performance and the relevance of the results that they offer.

It is therefore in the field of the evaluation of IRS and more specifically that of the contextual evaluation that our work falls. After a deep investigation around research and synthesis activities we realized that despite the abundant literature produced in this area dealing with both experimental results and methods that provide evaluation criteria and metrics of relevance, few of these methods are interested in the consideration of the context during the evaluation process. Our

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 31–45.
springerlink.com

contribution is guided by two main reasons: firstly the lack that we observed around the context-based methodologies for measuring the quality of information retrieval tools, this finding is reinforced by the work of [1] and [2]. And secondly by the requirement to which we are confronted recently after conducting work in the field of the consideration of context in information retrieval systems [3]; Where we have failed to find a contextual evaluation protocol to validate of our proposal. This work will therefore be a logical continuation of what has been done before, and a promising way to cover the process of the contextual evaluation of IRS.

This paper is organized as follows: we start first by giving a definition of the concept of context and its use in the field of information retrieval, we then present an overview of classical approaches for evaluating information retrieval systems and we focus on the limits and shortcomings faced by these approaches. In the next section we discuss our contribution by giving an overview of the contextual evaluation approach that we propose and describing its principle and its techniques. Finally, we present an experimentation of our approach to evaluate the performance of search engines Google, Yahoo and Bing, followed by a discussion of results and conclusion.

2 How Context Can Be Used in IR

2.1 Definition of Context

The context is not a new notion in computer science: from the sixties, operating systems, language theory and artificial intelligence already exploited this concept. With the emergence of information retrieval systems, the term was rediscovered and placed at the core of the debates without making subject of a consensus, clear and definitive definition. However, analysis of existing definitions in the literature leads to two conclusions:

"There is no context without context" [4]. In other words, the context does not exist as such. It is defined or it emerges for a purpose or precise utility.

"The context is a set of information. This set is structured, it is shared, it evolves and serves the interpretation" [5]. The nature of information and interpretations got from it depend on the purpose.

In information retrieval, the context is defined as "All cognitive and social factors as well as the user's aims and intentions during a search session", [6]. Generally speaking, the context includes elements of various natures that delimit the understanding, the application fields or the possible choice. The most commonly

cited elements concern the spatiotemporal data (location, time, date) or specific knowledge in relation to the studied area.

2.2 Use of Context in Information Retrieval

In information retrieval, context can be used at three different stages depending on the progress of the research process. Table 1, summarizes the potential to appeal to context in IR and presents the possible mechanisms of its use.

Table 1 Different opportunities to use the context in IR

Phase (Where)	Usage (How)	Examples of work (Who / What)
At the beginning of the search process	To solve the ambiguous terms problem in the query and improve the quality of the search tool results.	[7]: uses ontology with equivalence and subsumption relationships for extracting terms to be added to the initial query.
	To introduce the spatiotemporal constraints on search algorithms.	[8]: Proposes an algorithm for searching videos that capitalizes the context via spatio-temporal entities.
During the search process	To guide interactions with the system in order to make possible the real operating of the overall results once displayed.	[9]: Proposes a contextual dialogue platform that allowing a user to ask questions orally, and interact with IRS.
At the end of the search process	To guide the relevance feedback principle, with the idea of relying on results of the first search and the current context in order to reformulate the user query	[3]: Proposes a context based queries reformulation tool that use user's profiles to minimize the user intervention during reformulation.

3 Classic Evaluation of IRS, Principles and Limits

The classic evaluation of information retrieval systems is based on the performance of the systems in themselves; it is quantitative and is based on work done in the sixties at Cranfield (United Kingdom) on indexing systems [2]. Such approaches are mainly based on documents corpus, queries corpus, relevance judgments and evaluation metrics that are generally the recall and precision. [11]

3.1 The TREC and CLEF Evaluation Campaigns

The evaluation campaigns represent the current dominant model. Indeed, it is on the experience of the Cranfield tests that was based the NIST (National Institute of Science and Technology) to create the TREC evaluation campaign (Text REtrieval

Conference) in 1992. The TREC campaigns have become the reference in the evaluation of systems but we can also quote the CLEF Campaigns (Cross-Language Evaluation Forum) which specifically relate to the multilingual systems, the NTCIR campaigns on the Asian languages, and Amaryllis, specializing in French systems.

The TREC evaluation campaign is a series of annual evaluation of information retrieval technologies. The participants are usually researchers for large companies which offer systems and that want to improve it and academic research groups. The TREC is now considered as the most important development in experimental information retrieval. The main explored tracks are filtering, ad hoc task and question-answering. For 2010 TREC has focused on the following tracks: The blog, chemical IR, entity, legal, relevance feedback, and session tracks[1].

The CLEF campaign was launched in 2000 as an European project of evaluating information retrieval systems. The objective of the CLEF project is to promote research in the field of multilingual system development. This is done through the organization of annual evaluation campaigns in which a series of tracks designed to test different aspects of mono- and cross-language information retrieval are offered. The intention is to encourage experimentation with all kinds of multilingual information access – from the development of systems for monolingual retrieval operating on many languages to the implementation of complete multilingual multimedia search services. This has been achieved by offering an increasingly complex and varied set of evaluation tasks over the years. The aim is not only to meet but also to anticipate the emerging needs of the R&D community and to encourage the development of next generation multilingual IR systems [12]. CLEF 2009 offered eight main tracks designed to evaluate the performance of systems, the most important of these tasks are: Multilingual textual document retrieval, interactive cross-language retrieval, cross-language retrieval in image collections, intellectual property and log file analysis[2].

3.2 Limits of Classic Approaches for Evaluating IRS

Despite the popularity and recognition of these two evaluation campaigns that are TREC and CLEF. These approaches for evaluating information retrieval systems have some limits particularly with regard to the user consideration, the constitution of the queries corpus but also about the evaluation itself. To better identify the limits of classic approaches for evaluating information retrieval systems, we are based on the work of [1], [2] and [13]. A Synthesis of this work has allowed us to define six classes of problems, Table 2 summarizes these limits.

[1] TREC campaign web site: http://trec.nist.gov/
[2] CLEF campaign web site: http://www.clef-campaign.org/

Table 2 Limits of classical approaches for evaluating IRS

Nature of limit	Discussion
Absence of the user	– The notion of end user requires personal knowledge, experience and different search capabilities for which the classic IRS evaluation does not care.
	– [13]: classical evaluations do not take into account the context in which search is conducted, because they are not made in situations of actual use.
Relevance judgments	– The relevance judgments in TREC operate on a binary manner: a document is considered as relevant or irrelevant. Yet this is not always the case, some documents are more relevant than others who are also relevant.
	– [1]: The relevance considered in the classic evaluation of IRS is thematic, independent of context, of the research situation and interests of users.
Corpus of queries	– The problem with formulating queries in IR, transforms the task of search to a task of knows ask questions to these systems, because the differences are significant between what we think and what is interpreted.
	– [1]: in the batch mode of evaluation protocols, queries are assumed to represent alone the user. Consequently the direct user having made these queries does not form part of the collection.
Corpus of documents	– In the traditional corpus, a document is a text in itself, and the evaluation is made according to the number of documents found, but in general, a user is not looking for documents but information, and documents never contain the same amount of information.
Metrics	– [1]: The evaluation measures are not comprehensive and they do not permit evaluation of an operational search tool.
System Interaction	– [13]: The classical evaluation does not take into account the interactive nature of an information retrieval. An evaluation model that neglects the interaction is unrealistic and inappropriate for today IRS.

4 Detailed Presentation of the Proposed Approach

The aim of our evaluation approach is to consider the context of the user, of the query and that of the search tool during evaluation. It consists of three parts: evaluation of the search tool performance, evaluation of the results relevance compared to the query, and finally evaluation of the relevance by the user's judgments. Figure 1, summarizes the three evaluation levels and illustrates the link between the context type and the evaluation level.

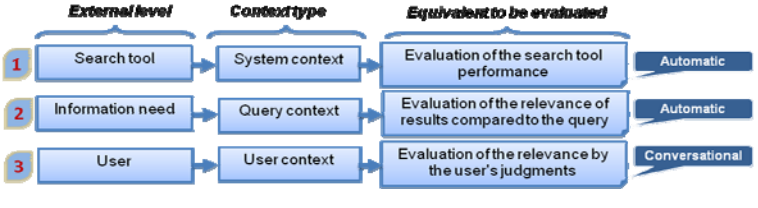


Fig. 1 Link between the context type and the evaluation level

4.1 Evaluation of the Search Tool Performance

This is the first component of our approach; the evaluation referred to this level is based on a number of criteria summarizing the problems generally encountered by users during a search session. The criteria that we have defined depend on the nature of the manipulated information, of the source of this information and finally of the mechanism used to retrieve this information. The values assigned to these criteria are automatically calculated by the system soon obtaining results provided by the search tool. The estimation of these values gives subsequently an overview of the quality of the search tool. These criteria are the following:

– *The redundant results:* This involves measuring the ability of the search tool to discard the redundant results. This means that the search tool should return only once the results coming from the same site but with different pages.

– *The dead Links: A* dead link is a link that leads to a page that no longer exists, In this case the browser returns in this case the error codes '404'. Evaluate this criterion consists to identify the ability of the search tool to detect and remove the dead links.

– *The parasites pages:* They include advertising pages and those that can identify only promotional links. These pages provide no useful information to the user and generally make false results. Their elimination depends to the quality of the algorithms used by search engines.

– *Response time:* This is the time consumed by the search engine to return the expected results. A short response time implies a good search tool performance.

4.2 Automatic Evaluation of the Results Relevance

This is the second part of our evaluation approach; this is the weighting, by increasing the number of terms, of the query words compared to the words of the returned documents. This includes choosing the weighted terms in the first time, then apply the formula that we propose.

4.2.1 Weighted Terms Choice, an Incremental Weighting

In an information retrieval process, queries reflect an information need and they are composed of one or more words. In such cases, groups of words in a query are often more semantically rich than the words that compose it taken separately, and can therefore better respond to what users expect.

We have chosen to define several hierarchal levels during weighting according to the number of words forming the query. Each level is composed of one or more words (a group of words) starting from the user's query. The incremental weighting of query terms instead of a classic weighting of each word separately allows to better taking into consideration the query context during the evaluation. For example, assuming that the query sent by the user is 'contextual evaluation of information retrieval systems', documents containing the group of words : 'contextual evaluation of information' or 'contextual evaluation' are certainly nearest to what

the user expect compared to those in which we find the words: 'contextual', 'evaluation', 'information', 'retrieval' or 'systems' taken separately.

4.2.2 Relevance Calculating, a Contextual Formula

Once the groups of words to be weighted are defined, it comes to assigning a weight that determines their importance in the document. We have therefore developed a weighting formula that takes into account the context of the query in terms of number of words composing it. This formula is inspired from the TF IDF weighting [14] to which we added two dimensions; the document length and the hierarchy of words groups according to the length of the query. So, it is incremental and is defined as follows:

$$W(R,D) = \sum_{R' \in R} \left[\frac{W(R',D)}{length\,(D)}\right] * \left[\frac{length(R')^2}{length\,(R)}\right] * \log_2\left[\frac{TNRD}{NDWDR'}\right]$$

With:

– R: The set of query terms.
– R': The terms of the words group to weighted.
– W (R',D): The frequency of the word group R' in the document D.
– Length (R): the length the query.
– Length (R'): the length the query words group to weighted.
– Length (D): Length of the document.
– TNRD: Total number of returned documents.
– NDWGR': Number of documents containing the words group R'.

4.3 Evaluation of the Relevance by the User's Judgments

The interest that a user gives to information depends heavily on individuals and context of use. Information will therefore be important for a given user in a given context. Based on this principle and to allow consideration of the user's judgments during the evaluation, we use an adaptation of our proposal [3] which is to model the context of the user via their profile. This adaptation requires a redefinition of the concepts of static and dynamic context to make them usable for evaluation.

4.3.1 Static Context

These are the personal characteristics of the user that can influence the evaluation context. This information is to be recovered during the first connection to the system. For this purpose we have identified four categories of information relating to the static context, this information is summarized in:

– Connection parameters: e-mail and password.
– Personal characteristics: name, country, language,...
– Interests and preferences: domains, specialty,...
– Competence expertise level: profession, level of study,...

4.3.2 Dynamic Context

In order to optimize the reuse of the user's judgments, this second component of context aims to associate the relevance judgments with the user's context. The principle is as follows: at the end of each search session the recovery of the dynamic context is performed and this by allowing users to express their judgments of relevance regarding to the documents returned by the search tool. This judgment is to vote on a scale from 0 to 5, where 0 corresponds to a document completely useless or off-topic, 5 corresponding to a document that responds perfectly to the asked query. The evaluation is activated automatically whenever the user expresses a judgment by recalculating the relevance score of the considered result.

5 Application of the Proposed Approach to the Evaluation of Search Engines

To prove the applicability of the proposed approach, we will use it for evaluating search engines. Our choice was fixed on three search engines (Google, Yahoo and Bing). This choice is motivated by their popularity in the web community on the one hand, and by the high degree of selectivity that they provide on the other hand.

We therefore propose to establish a system that conducting an open search on the web, and perform by following the evaluation of the results returned by each search engine. To this end we use the three levels of the contextual evaluation approach that we have proposed. This system should allow:

- Make the same queries set to the three search engines (Google, Yahoo, Bing).
- Retrieve the results returned by each search engine;
- Check the informational content of all the resulting pages;
- Capture the user's static and dynamic context for the current search session, and used them for the evaluation of the results by the user's judgment;
- Measuring the relevance degree of results returned by each engine by the application of the proposed contextual formula.
- Diagnose performance, characteristics and behavior of each search engine by taking into account its context accordance to the third level of our approach.
- Coupling of the relevance scores obtained in the three evaluation levels for each search engine and thus obtained the final score.

The system consists of two main modules: a first module for managing interactions between the user and the search engine (identification and search), and a second which covers the three evaluation levels described in our proposal. These two modules are closely interrelated in the sense that the outputs of a module are the inputs of the other. We present in what follows, modules components the system and we illustrate the functionalities offered by each of them.

5.1 Managing of Users / Search Engine Interactions Module

A preliminary phase to this evaluation is absolutely necessary, it is necessary to recover the user's information need and then interrogate the search engine to retrieve results to be evaluated. The managing of users/search engine interactions module supports all interactions from the connection to the system until the results deliverance. It takes care capturing of the user's static context, managing of its identification, he also manages the transmission of the user request to the search engine and retrieval of results, and finally, it communicates these results to the evaluation module. This module consists of two complementary processes, Figure. 2, shows the operating principle of this module.

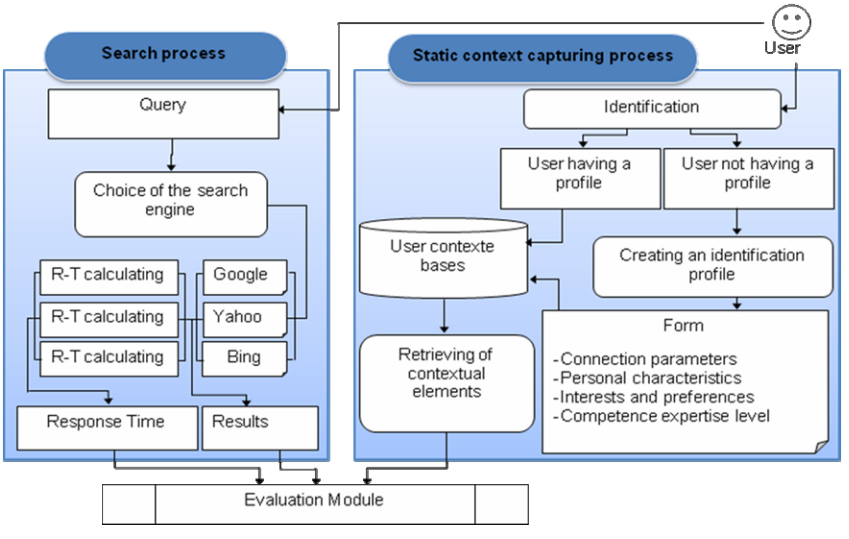


Fig. 2 Managing of users/search engine interactions module.

5.1.1 The Static Context Capturing Process

The static context previously defined during the presentation of our approach is represented by the user profile. The user profile data can be indicated by the user himself, learned by the system during use or indicated by selecting an existing profile created by experts.

In our case, we construct the user static context at the first connection to the system. This construction is done by asking the user to fill the four categories of information defined previously. The categorization of users has the advantage of having typical information with the opportunity to refine it as and when. Once the identification made the user can conduct open research on the web.

5.1.2 The Search Process

We opted for a system that offers an open search on the web using the following principle: after connecting to the system, the user expresses his information need as a query. The research process therefore takes as input the query and the search operation is initiated by running in parallel the nucleus of each search engine with as only parameter the user query. The obtained result is finally communicated to the user and the evaluation module.

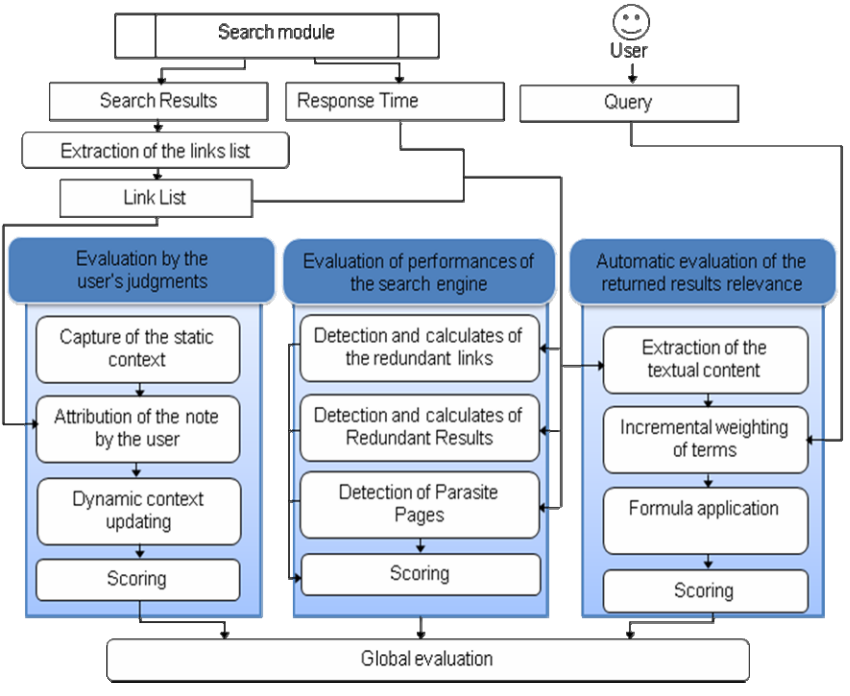


Fig. 3 Summary of the evaluation approach applied to search engines

5.2 Contextual Evaluation Module

To precede with the evaluation of the three search engines, the system retrieves the results of each of them and performs their analysis. The contextual evaluation module consists of three processes representing the three evaluation levels of the proposed approach. The following algorithm illustrates the operating principle of the evaluation module, and Figure 3, summarizes our evaluation approach applied to search engines.

```
Algorithm
    For each search engine Do
        For each query Do
            Calculate the response time
            For each resulting webpage Do
                Extract all the url of the webpage
                For url(i) = 1 To 20 Do
                    If url(i) = url(i+1)  Then
                        Nbr Redundant Link ++
                    End If
                End For
                Open the corresponding page for each url
                If the returned error code is 'http 404' Then
                        Nbr Dead Links ++
                End If
                For j = 1 to length (Q) Do
                    Calculate the frequency of the word(j) in the resulting page
                    If frequency of the word(j)=0  Then
                        Nbr Parasite Page ++
                    End If
                End For
            End For
              Update the performance evaluation of search engine
            For k = 1 to length (Q) Do
                G = G + Word (K)/Concatenation
                Apply the formula1 for the words group 'G'
            End For
              Update the evaluation of the relevance of results
                  Capture of the static context
                  Capture of the dynamic context
                  Retrieve the relevant user judgments
              Update the evaluation by the user's judgments
        End For
    End For
End
```

6 Results and Discussion

6.1 The Used Protocol

To measure the contribution of our approach to the search engines evaluation, we use an extension of the evaluation scenario proposed in [15].The evaluation was conducted with the help of 24 students from the second year license STIC (Science and Technology of Information and Communication) at the Mentouri Constantine University, playing the role of users. The goal was not to make an evaluation by experts but by a basic public, reasonably familiar with search engines. 6 topics were chosen, to reflect diverse fields of use. These topics are: News, Animals, Movies, Health, Sports and Travel. Each topic was assigned to a group of 4 students who chose freely 5 queries. For example, for the sports topic, the chosen queries were as follows: {World Cup 2010, France cycling tour, Formula 1 racing cars, Famous football players, Roland-Garros tournament}.

Queries were submitted to different search engines, and the first two pages containing the 20 results were archived for each query and each search engine. In total, 1800 'url' ware retrieved (6 topics x 5 queries x 20 results x 3 search engine)

and organized in the form of triplet (Query, url, page content). Finally the set of triples has been communicated to the system for analysis and evaluation.

6.2 Performance of Search Engines

Regarding the dead links, Table 3 shows that the rate of dead links is low, this is explained partly by the fact that some web site do not return the error code 404 'Page not found' when the page no longer exists, but a normal HTML page with an ad hoc message, which cannot be interpreted as an error only by a human reader. We note also that 71% of dead links returned by Yahoo and 79% of those returned by Google are caused by the Amazon which, for unknown reasons, returned an error code during the experiment. Finally, Bing has got the best score with only 1.67% of dead link.

Table 3 Search engines performance evaluation

Search engines	Performance			
	Dead links	Parasites pages	Redundant results	Average response time
Google	2,03%	5,30 %	4,04%	0,17 Sec
Yahoo	2,13%	10,19 %	4,81%	0,21 Sec
Bing	1,67%	8,64 %	5,32%	0,22 Sec

In terms of parasites pages, which are essentially links that referring to commercial web sites. We notice that search engines have different strategies to exclude the parasite pages. Among the commercial sites that appear several times we notice two companies: Amazon and E-Bay. Overall, it is Google that returns the fewest links to commercial sites with 5.30%.

Concerning redundant results, we find that the ability of the three search engines to eliminate them varied according to the type of queries. We also note that the majority of redundant links returned by Google and Yahoo comes from the use of Wikipedia web site. Of the 20 analyzed results, Google returned 4.04% redundant links whose 80% from Wikipedia, and Yahoo 4.81% redundant links whose 78% from Wikipedia.

Finally, the average response time depends heavily on internet connection speed and the machine power. For this reason and to ensure homogeneity when calculating this criterion, all queries have been tested on the same machine with the same internet speed. The obtained results show that the average response time is almost identical in the three search engines. However, we note that Google top the list with 0.17 seconds, this can be explained by the power of the PageRank algorithm used by this engine.

6.3 Relevance by the User's Judgments

We are interested in the relevance judgments given by the user for the first result returned by each search engine (R@1). The latter is of particular importance, since it is the closest link clicked by users. The 24 students also expressed their relevance judgments for 5, 10, 15 and 20 first retrieved documents (R@5, R@10, R@10, R@15, R@20). At each level of relevance, a note of 0-5 was assigned by each student. 0 corresponding to a document completely useless or off-topic, 5 corresponding to a document responding in a perfect way to the question. Table 4, shows the obtained scores.

Table 4 Evaluation of the relevance by the user's judgments

Relevance level	Search engines		
	Google	Yahoo	Bing
R@01	3,15	2,92	2,70
R@05	2,79	2,14	2,58
R@10	2,34	2,51	2,16
R@15	2,00	1,83	1,72
R@20	1,91	1,77	1,69

The overall scores obtained by each search engine for the 20 results are extremely low, since no engine reaches the average note of 2.5 at R@20. The search engine that had the best note of 1.91 is Google. The situation is remarkably improved if one considers only the first result R@1; the three search engines are exceeding the average.

6.4 Results Relevance according to the Query

Using our formula, we calculate the relevance of the first 20 returned results regarding to each of the 30 queries, and that for the three search engines. A note average for each group of 5 queries in the same topic was calculated and the obtained score was rounded to a note on 10. The overall results are summarized in Table 5.

Table 5 Evaluation of the results relevance according to the query

Queries category	Search engines		
	Google	Yahoo	Bing
R_{01} - R_{05} (News)	6,91	6,77	6,19
R_{06} - R_{10} (Animals)	5,25	6,13	5,87
R_{11} - R_{15} (Movies)	5,72	5,13	5,67
R_{16} - R_{20} (Health)	4,98	4,83	4,66
R_{21} - R_{25} (Sports)	5,93	5,89	5,16
R_{26} - R_{30} (Travel)	6,19	6,09	6,10

The analysis of the obtained results show that the Google search engine ranks first in terms of results relevance according to the query, and that for the 5 query categories of the 6 available categories. This finding may be explained by a possible match or an unintended complicity between the formula that we proposed and the mechanism used by Google to rank results. We also note that the scores of the 'health' category are below average for the three search engines, this is due to the fact that the queries in this category contain few of words, which decreases terms for which we calculate the number of occurrence and thus weaken the final score.

7 Conclusion

In this paper we are interested in proposing a new contextual approach for evaluating information retrieval tools. Our main contribution consists of the consideration of context during the evaluation at three complementary levels. First the context of the system is considered by estimating the ability of the search tool to eliminate the dead links, redundant results and parasites pages. In a second level our approach takes into account the query context based on an incremental formula for calculating the relevance of the returned results according to the user's query. The last level of the approach takes into consideration the user's judgments via his static and dynamic context. Finally, a synthesis of the three levels of contextual evaluation was proposed.

The application of the proposed approach to the search engines evaluation was used to demonstrate its applicability for real research tools. This study which is certainly far from exhaustive, nevertheless gives a snapshot of the search engines performance and the relevancy of results that they return. Finally, this study paves the way for diverse perspectives; particularly in terms of enlarging the application field of the realized research. It would be interesting to test the proposed approach to evaluate personalized search tools and enrich the obtained results with search engines.

References

[1] Tamine, L., Boughanem, M., Daoud, M.: Evaluation of contextual information retrieval effectiveness: overview of issues and research. Journal of Knowledge and Information Systems 24(1), 1–34 (2010)

[2] Menegon, D., Mizzaro, S., Nazzi, E., Vassena, L.: Benchmark evaluation of context-aware Web search. In: Workshop on Contextual Information Access, Seeking and Retrieval Evaluation, Toulouse, France. Springer, Heidelberg (2009)

[3] Bouramoul, A., Kholladi, M.K., Doan, B.L.: PRESY: A Context based query reformulation tool for information retrieval on the Web. Journal of Computer Science 6(4), 470–477 (2010), ISSN 1549-3636

[4] Brézillon, P.: Making context explicit in communicating objects. In: Kintzig, C., Poulain, G., Privat, G., Favennec, P.-N. (eds.) Communicating with Smart Objects, ch. 21, pp. 273–284. Kogan Page Science, London (2003)

[5] Winograd, T.: Architectures for context, human-computer interaction, vol. 16, pp. 402–419. L. Erlbaum Associates Inc., Hillsdale (2001)
[6] Belkin, N., Muresan, G., Zhang, X.: Using User's Context for IR Personalization. In: Proceedings of the ACM/SIGIR Workshop on Information Retrieval in Context (2004)
[7] Navigli, R., Velardi, P.: An analysis of ontology-based query expansion strategies. In: Proceeding of the Workshop on Adaptive Text Extraction and Mining, Dubrovnik – Croatia (2003)
[8] Chen, X., Jia, K., Deng, Z.: A Video Retrieval Algorithm Based on Spatio-temporal Feature Curves and Key Frames. In: Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, Kyoto, Japan, pp. 1078–1081 (2009)
[9] Rosset, S., Galibert, O., Illouz, G., Max, A.: Interaction et recherche d'information: le projet Ritel. Traitement Automatique des Langues (2006)
[10] Lin, H.-C., Wang, L.-H.: Query expansion for document retrieval based on fuzzy rules and user relevance feedback techniques. Expert Systems with Applications 31(2), 397–405 (2006)
[11] Daoud, M., Tamine, L., Boughanem, M.: A contextual evaluation protocol for a session-based personalized search. In: Workshop on Contextual Information Access, Seeking and Retrieval Evaluation, Toulouse, France. Springer, Heidelberg (2009)
[12] Peters, C.: What Happened in CLEF 2009. In: Peters, C., Di Nunzio, G.M., Kurimo, M., Mostefa, D., Penas, A., Roda, G. (eds.) CLEF 2009. LNCS, vol. 6241, pp. 1–12. Springer, Heidelberg (2010)
[13] Chaudiron, S., Ihadjadenem, M.: Quelle place pour l'usager dans l'évaluation des SRI? In: Couzinet, V., Regimbeau, G. (eds.) Recherches récentes en sciences de l'information: Convergences et dynamiques. Actes du colloque international MICS LERASS, Toulouse, March 2002, pp. 211–232 (2002)
[14] Soucy, P., Mineau, G.W.: Beyond TFIDF Weighting for Text Categorization in the Vector Space Model. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland (2005)
[15] Véronis,J.: Etude comparative de six moteurs de recherche. Université de Provence (2006),
http://sites.univ-provence.fr/veronis/pdf/2006-etude-comparative.pdf

Quantitative Verification of Non-functional Requirements with Uncertainty

Carlo Ghezzi[1] and Amir Molzam Sharifloo[2]

[1] Dipartimento di Elettronica e Informazione, Politecnico di Milano,
P.zza Leonardo da Vinci 32, 20133 Milano, Italy
e-mail: carlo.ghezzi@polimi.it
[2] Dipartimento di Elettronica e Informazione, Politecnico di Milano,
P.zza Leonardo da Vinci 32, 20133 Milano, Italy
e-mail: molzam@elet.polimi.it

Abstract. We focus on non-functional requirements, such as those concerning reliability, performance, or cost and examine how to support the transition from requirements to design models that can be analyzed formally in quantitative terms. We assume that the initial description is given in behavioral terms, using annotated UML Sequence Diagrams. Annotations are used to express environmental assumptions, which are subject to uncertainty, in probabilistic terms. We also assume that a set of requirements is expressed via Structured English statements, which provide predefined patterns to support specification of common probabilistic properties. We discuss how sequence diagrams can be automatically translated into formal models that support software engineers in reasoning about the application being developed. In particular, requirements are transformed into appropriate logic statements while sequence diagrams are translated into Markov models, which can then be analyzed by using probabilistic model checking.

1 Introduction

The complexity of modern software systems has grown enormously in the past years. Moreover, they are increasingly used in critical applications that require high dependability. In addition, users are always demanding for new features and better *quality of service* (QoS). The traditional approach to achieving quality relies on software models that capture the relevant aspects of the system being designed. Models can be analyzed to predict whether the future system satisfies the requirements prior to the system implementation. By following a model-driven approach and by applying verification at the model level, software engineers can anticipate detection of possible flaws that would otherwise become part of the implementation. This is indeed the main driving force behind *model-driven engineering* (MDE) [3].

Very often the system under development will live in an environment that cannot be fully anticipated at design time. On the other hand, the properties of the

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 47–62.
springerlink.com

environment influence requirements satisfaction. For example, some assumptions concerning the way the application will be used (usage profiles), which may be hard to know in advance, may ultimately affect the application performance.

In general this is true for non-functional requirements, which depend on uncertain and difficult to predict environmental data. Uncertainty is normally dealt within probabilistic terms. For example, one may assume a certain intensity probability distribution, and based on that one may be able to prove that the design satisfies the requirements in terms of response time.

This paper focuses on requirements verification in the initial design phases of software development. It focuses on non-functional requirements and high-level design models. It shows how model checking can be used to assess models against the requirements. More precisely, we assume that initially the software engineer describes the desired system functionalities through behavioral models that represent high-level scenarios, specified via UML Sequence Diagrams (SDs). Quantitative stochastic annotations are used to decorate SDs with assumptions on the external world. For example, they may describe the probability that a certain branch of the scenario can be selected, due to the expected usage profile. A probability may also be attached to an asynchronous message, to state the probability that invocation of an external service terminates successfully. It is also possible to attach an average duration to the transmission of a message or to the execution of certain operation. Concerning QoS requirements (in this paper, our focus is on reliability, performance, and cost), we assume that the software engineer specifies global system requirements by using Structural English statements, which support predefined patterns to specify common properties, as described by [8]. These statements are then transformed into probabilistic logic formulae that can be verified to hold on the model.

To achieve verification, the paper presents a translation step from annotated SDs into formal models on which requirements can be verified. Because of the nature of the properties, we wish to express, analyze and translate it into Markov models: DTMCs (Discrete Time Markov Chains), CTMCs (Continuous Time Markov Chains), and Reward DTMCs [10]. Once the translation is performed, probabilistic model checking algorithms can be applied to verify if the requirements for a given behavioral system description hold on the corresponding Markov models.

The paper is structured as follows. Section 2 describes the proposed framework. Section 3 focuses on the transformations from SD diagrams into Markov models. A running example is presented in Section 4. Finally Section 5 gives an overview of related work and Section 6 discusses future work.

2 The U-MarMo Framework

This section describes the proposed framework for quantitative verification of non-functional requirements and introduces the notations of the source and target models used in transformations. *U-MarMo* (*UML to Markov Models*) consists of a model-to-model transformation step followed by model checking (e.g., [13]) to formally verify non-functional requirements on UML diagrams. Currently

U-MarMo supports SDs decorated with quantitative annotations and generates Markov models. DTMCs and CTMCs are used to verify reliability and performance properties. A variant of DTMCs is used to verify cost properties, such as energy consumption.

According to Fig. 1, initially developers provide system descriptions in the form of annotated SDs. They also provide non-functional requirements written in Structured English using predefined patterns [8]. SDs are annotated with probabilities to support reasoning about reliability via transformation into DTMCs. SDs annotated with performance rates may be translated into CTMCs. SDs annotated with cost parameters can be transformed into Reward DTMCs, a variant of DTMCs used for reliability verification. Annotations and transformation are described later in Sections 2.1 and also 3.

To annotate SD diagrams, we use the MARTE profile. For more details about UML SDs and the MARTE profile, please refer to [20] and [12].

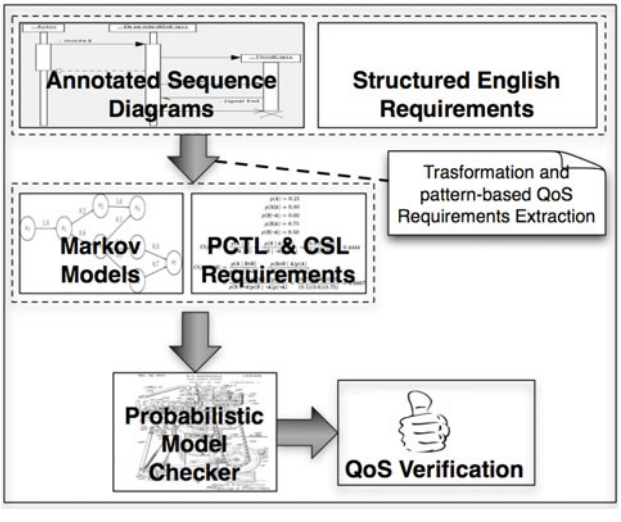


Fig. 1 The U-MarMo Framework

2.1 Sequence Diagrams, Semantics, and Annotation

U-MarMo supports initial high-level system descriptions given in terms of SDs. Precisely, U-MarMo supports the following building blocks: *lifelines*, *messages*, and *combined fragments*. It also supports the following types of messages: *synchronous*, *asynchronous*, and *reply*; they are indicated by a line with solid arrowhead, a line with an open arrowhead, and a dashed line with an open arrowhead, respectively. Messages play a major role; they are used to represent both communication and computation. In fact, we use *self-messages* to indicate the execution of internal actions by a lifeline. While inter-object messages stand

for a communication. Fig. 2(a) illustrates these various message types. Please note that if an object sends a synchronous message, it remains blocked until it receives a reply message. Instead, when an object sends an asynchronous message, it continues with the rest of the actions it is expected to perform. A reply message is a kind of asynchronous message, as far as the issuer is concerned.

As for semantics, every lifeline describes a parallel process (a state machine). The state machines representing the various lifelines evolve continuously and obey partial order semantics, which includes two rules: (1) the actions of a lifeline are sequentially ordered from top to bottom, (2) a message cannot be received before it is sent.

Sets of messages can be grouped together in combined fragments, graphically represented by a box. The official specification of UML comprises many different types of combined fragments. Hereafter we focus on the four major fragments: (1) Alternative, (2) Option, (3) Loop, and (4) Parallel.

Mutually exclusive choices between two or more sequences of messages are represented using *Alternative*. Each Alternative contains a set of operands (each of which is a group of messages) separated by a dashed line. Each operand is associated with a condition and is executed if the condition evaluates to true (Fig. 2(b)). Note that the condition of the Alternative is evaluated only once when all the lifelines participating in it have reached the Alternative. In fact, we follow the proposal by Alur [2] and apply a synchronous approach, which is also used for the other fragments (Option and Loop) in which there are conditions.

The *Option* fragment is used to model a sequence that occurs if and only if a certain condition evaluates to true (Fig. 2(c)). SDs represent iterating sequences of messages through *loops*. A Loop is associated with a condition and the sequence of messages included in the fragment is executed as long as the condition evaluates to true (Fig. 2(d)). The *Parallel* fragment represents parallel computations (Fig. 2(e)).

The elements of SDs are annotated using the MARTE profile. We annotate all messages of a diagram with two MARTE properties: execTime and prob. The former – a non-negative real number – represents the mean time for transmission. The latter represents the probability that message transmission is successfully performed. Note that since self-messages represent an internal operation, these properties represent duration time and operation reliability, respectively. For cost properties, there are different choices depending on the type of cost: message size, power, and energy are examples that can be used in the MARTE profile. All of them are represented by a real number and can be seen as cost of resource usage. In the rest of this paper, we implicitly focus on energy consumption.

Combined fragments, except Parallel, are annotated with execution probabilities. A probability attached to an Option indicates the likelihood that the Option is chosen. Similarly, each loop is annotated with a probability, which expresses the probability that the loop may iterate. Since Alternative includes more than one operand, each operand is annotated with an execution probability.

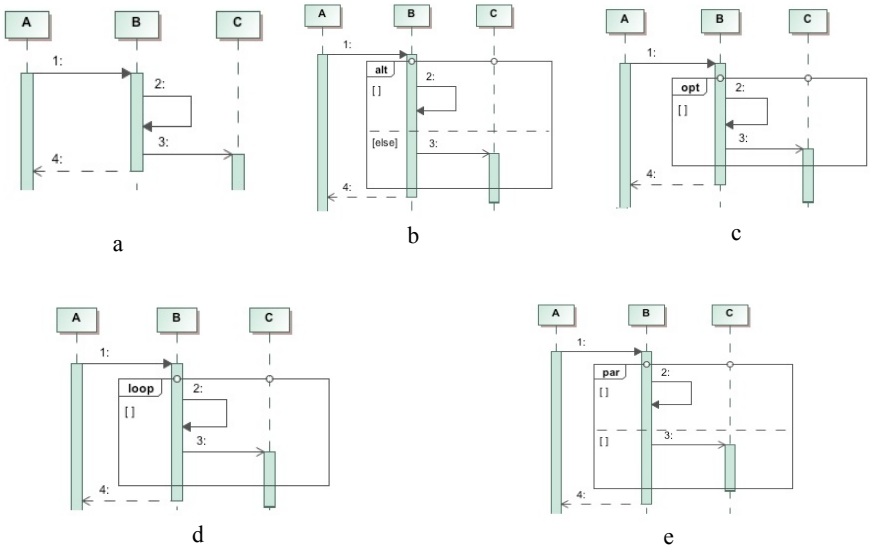


Fig. 2 Sequence Diagrams

2.2 Using Property Patterns to Specify Non-functional Requirements

Our framework uses the probabilistic pattern system ProProST introduced by Grunske [8] to specify non-functional requirements using Structured English. Although there exist other pattern systems to specify non-functional requirements [19], ProProST is the one specifically presented for probabilistic properties. That is the reason we chose ProProST. Moreover, the property specified via ProProST can be translated into the logic specification languages PCTL or CSL automatically. Examples will be given in the case study we will discuss next.

For cost properties, we use a variation of Grunske's patterns. Since cost properties are expressed as real numbers, we replace the probabilities with real numbers (which represent cost) in the grammar of ProProST. The formulae we obtain by the translation are cost/reward formulae as used by the PRISM model checker for cost/reward properties [10].

2.3 Target Models and Requirements Specification

To support formal analysis of requirements, we generate Markov models that are amenable to model checking. We generate Discrete-Time Markov Chains (DTMCs) for reliability analysis and Continuous-Time Markov Chains (CTMCs) for performance. DTMCs with rewards are used for cost analysis. For space reasons, we only provide a sketchy introduction to DTMCs and CTMCs and to the

languages (PCTL, CSL) in which properties may be expressed and analyzed. The reader may refer to [10] for details.

A Markov Chain can be viewed as a state machine, where transitions are annotated. Annotations are probabilities for DTMCs and rates for CTMCs. For example, the DTMC in Fig. 3 describes a system where in the initial states two actions may be chosen (A, with probability 0.7 and B, with probability 0.3). Execution of A may then succeed with probability 0.9 – leading to state C – or fail –leading to state D. Similarly, B may succeed with probability 0.99 or fail with probability 0.01. CTMCs label transitions with real numbers, representing rates instead of probabilities. DTMCs with reward instead label transitions with real numbers representing costs.

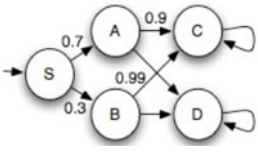


Fig. 3 Sample DTMC model

As for the property languages – PCTL, CSL, and cost/reward formulae – they all belong to the family of temporal logic languages. For example, in PCTL we can express a property concerning the reachability of a failure (or success) state. Similarly, in CSL we can express properties on the response time distribution of a certain transaction. Examples will be given later in the paper.

Annotated SDs are the main notations we use to formally describe system models that allow us to reason on requirements specification during design. As mentioned before, each message of the diagram is annotated with reliability, performance, and cost parameters. These values are elicited by consulting domain experts or simply based on previous experience with similar systems. Moreover, each combined fragment (except for Parallel) is annotated with execution probabilities.

3 From Sequence Diagrams to Markov Models

According to the semantics of SDs outlined in Section 2, each lifeline can be represented by a state machine. The overall state machine that represents the entire state machine can be obtained by performing a parallel composition of the state machines representing the different lifelines, under the constraints imposed by the messages exchanged between them. If we also consider the annotations on the transitions, the resulting state machines can be either DTMCs or CTMCs depending on the type of annotations we consider. The model transformations from SDs to DTMCs and to CTMCs, discussed in this section, are exactly based on these semantic foundations. For example, Fig. 4(a) shows the state machine for each lifeline of the SD in Fig. 2(a), while Fig. 4(b) illustrates the corresponding

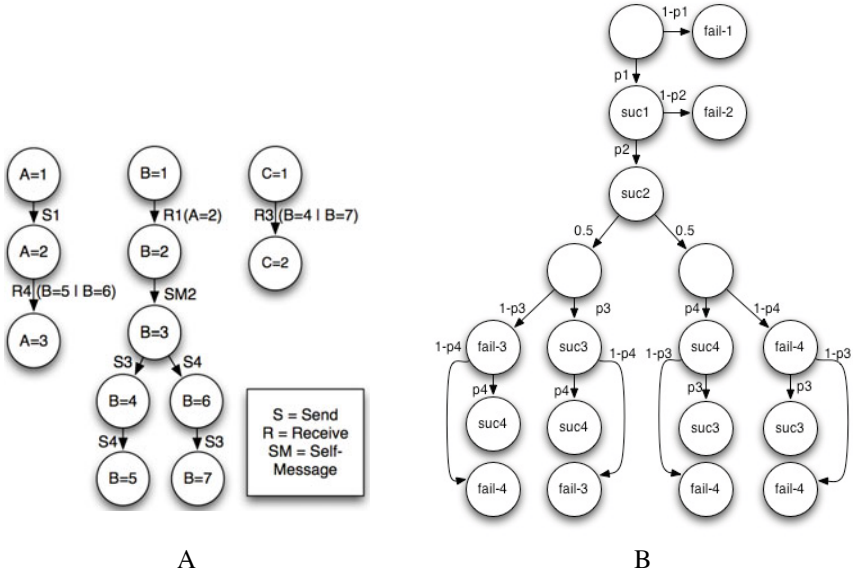


Fig. 4 (a) The state machines for the lifelines in Fig. 2(a) - (b) The corresponding DTMC for the SD in Fig. 2(a)

```
RetrievePerformableActions(SD sd){
        define actions as a list of Action;
        foreach(lifeline[i] in sd.lifelines)
        if( ! lifeline[i].IsFailed() and ! lifeline[i].IsFinished() ){
                actions.Add(RetrievePerformableActions(lifeline[i]));
        }
}

RetrievePerformableActions(Lifeline l){
        define actions as a list of Action;
        while(){
                switch(l.current_action.type){
                        case "RECEIVE":
                                if(!IsAlreadySent(l.current_action))
                                        return actions;
                                else
                                        l.moveToNext();
                        case "SEND_ASYN":
                                actions.add(l.current_action);
                                l.moveToNext();
                        case "SELF-MSG" or "SEND_SYNC":
                                actions.add(l.current_action);
                                return actions;
                }
        }
}
```

Pseudo-code 1. Retrieving performable actions

DTMC constructed through composition of the state machines. Note that S stands for Send, R stands for Receive, and SM for Self-Message. Unlike Send and Self-Message, Receive has a condition to make sure that the message is sent before. Let us consider the lifeline A. As shown in Fig. 4(a), the lifeline A has three states and two transitions. The first transition says that message 1 is sent. The second transition expresses that the message 4 is received only if the lifeline B is in the state 5, 6 or 7, which are possible states after having the message 4 sent. Also note that the state machine for lifeline B has a branch, which represents the fact the completion of sending the message 3 may occur either before or after the completion of sending of the message 4. The two branches describe exactly these two possibilities. The state machines of the other lifelines can be explained in the same manner. Although the semantics behind the transformation can be seen as composition of state machines, we do not create any state machine in the transformation. Instead, we directly transform SDs into Markov models. The remainder of the section describes the transformation in details.

Regardless of the transformation, first we need to find the order in which actions are performed (which in our case corresponds to finding the order in which messages are transmitted) by iteratively performing a search in the diagram to find performable actions. We start in an initial state in which lifelines are initiated, and then we find performable actions (self-message or message) and transform them into transitions of target Markov models.

Pseudo-code 1 illustrates the algorithm to extract performable actions. For simplicity, let us assume that the SD does not include any combined fragment. To find the actions, the method goes through each lifeline and checks the upcoming actions that the lifeline is supposed to perform. If the action is a self-message, it is performable without any condition. In case it is a receive action, it is necessary to check whether the sender lifeline has transmitted the message. If it has done so, receiving is performed and also the next action is checked. If action is sending a message, the type of message is considered. If it is synchronous, it is added as a performable action. If it is asynchronous, not only it is added to the list of performable actions, but also the next performable action is sought. The reason is that sending an asynchronous message does not block a lifeline, so it can continue immediately with the next action.

3.1 Transforming SDs into DTMCs

An SD is transformed into a DTMC to analyze satisfaction of reliability requirements. The actions on the source SD are annotated with their success probability P (1-P is the probability of failure). Pseudo-code 2 describes how to map an SD onto DTMC. The method Transform invokes the method *RetrievePerformableAction* (discussed earlier) to extract actions given the execution locations of each lifeline. Each action is transformed into two transitions in the DTMC, representing success and failure. If an action is performed correctly, the corresponding lifeline in the SD moves to the next action and a success transition is added to the DTMC. Otherwise, the receiver lifeline involved in the action fails and a failure transition is added to the DTMC (Fig. 5(a)). In

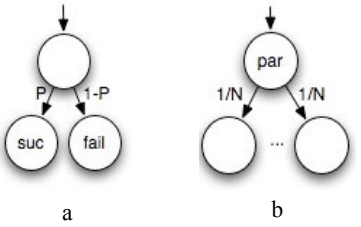


Fig. 5 Transformation rules for DTMCs

pseudo-code 2, the methods Create-Success and Create-Failure are responsible for adding these transitions to DTMC.

After performing any action, the DTMC transitions from the current state to a new state. If there is more than one performable action, we add new transitions to represent the interleaving. For instance, Fig. 5(b) shows that N transitions are added to the DTMC. Each of these transitions assumes that one particular action is performed before the others. The probability of each transition equals 1/N, N being the number of performable actions. Fig. 4(b) presents the DTMC for the SD of Fig. 2(a).

For space reasons, we explicitly omitted the treatment of combined fragments Option, Alternative, Loop, and Parallel in the SD to DTMC translation. Their treatment, however, is rather straight-forward and does not introduce new conceptual problems.

3.2 Transforming SDs into CTMCs

An SD is transformed into a CTMC to verify performance requirements. As for translation into DTMC, we start with an initial state, find performable actions, and then transform them into states and transitions of a CTMC. The main difference here is that transitions of a CTMC are labelled with rates instead of probabilities. We again refer to the method *RetrievePerformableActions* to find actions, but the rest of the transformation is different. Each action is mapped onto a transition with the corresponding performance rate. Fig. 6(a) shows the corresponding CTMC for the SD in Fig. 2(a). After performing any action, CTMC transitions to a new state. Similarly to DTMC, if more than one performable action can be chosen, we explore all the interleavings, but we do not need to add new transitions.

In the case of an Option (Fig. 2(c)) (with a probability associated with the branch), we use a suitable rule to combine the probabilities and the rates of transitions. The rule is that both the first action inside the Option (message 2) and the first action after the Option (message 4) are performable. Thus when an Option is reached, two different traces may be executed. To map onto CTMC, we go back to the last transition visited (with rate 'r1' in the Fig. 6(b)) and divide it into two transitions with different rates. The first transition has rate (r1*opt), where r1 stands for the action rate of message 1 and opt is the probability that the option

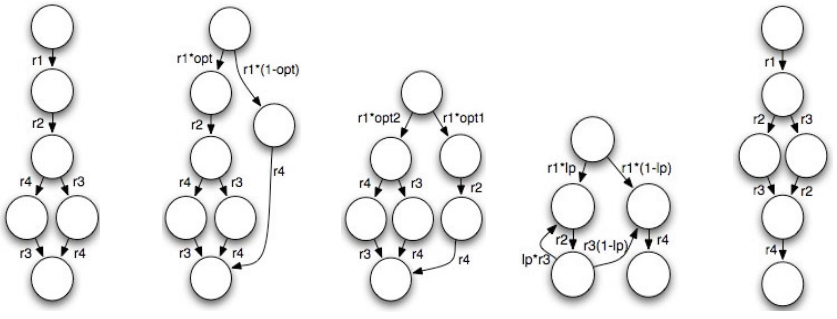
```
TransformSDtoDTMC(SD sd){
  define dtmc as DTMC;
  define initial_State as State;
  define actions as list of Action;
  dtmc.add(initial_State);
  Transform(sd, dtmc, initial_state, actions)
}

Transform(SD sd, DTMC dtmc, State current_state, Action[] actions){
  while(!sd.IsFinished()){
   actions.Add(RetrievePerformableActions(sd));
   if(actions.size==0)
    return; //the procedure terminates
   if(actions.size==1){
    action = actions.Get[0];
    Create_Success(current_state, action, actions, sd, dtmc);
    Create_Failure(current_state, action, actions, sd, dtmc);
   }
   else
    if(actions.size>1){
     float branchProb = 1/actions.size;
     foreach(action in actions){
        State middle_state = new State;
        middle_tra = new Transition(current_state, middle_state, branchProb);
        dtmc.add(middle_state);
        dtmc.add(middle_tra);
        Create_Success(middle_state, action, actions, sd, dtmc);
        Create_Failure(middle_state, action, actions, sd, dtmc);
     }
    }
  }
}

Create_Success(State c_state, Action action, Action[] actions, SD sd, DTMC dtmc)
{
    State s_state = new State();
    Transition s_tra = new transition(c_state, s_state, action.prob);
    dtmc.add(s_state);
    dtmc.add(s_tra);
    SD sd1=sd.Copy();
    sd1.MoveOver(action); // we update current action for each lifeline involved in action
    actions.Remove(action);
    Transform(sd1, dtmc, s_state, actions);
}

Create_Failure(State c_state, Action action, Action[] actions, SD sd, DTMC dtmc)
{
    State f_state= new State();
    Transition f_tra = new Transition(c_state, f.state, 1-action.prob);
    dtmc.add(f_state);
    dtmc.add(f_tra);
    SD sd1=sd.Copy();
    sd1.Fail(action.receiver_lifeline); // the receiver lifeline is marked as failed
    Transform(sd1, dtmc, f_state, actions);
}
```

Psuedo-code 2. The basic algorithm to transform SDs into DTMCs

evaluates to true. The second has a rate equal to r1*(1-opt), which corresponds to the path when Option is not executed. Fig. 6(b) shows the output CTMC for the SD in Fig. 2(c).

The transformation of a Loop is similar to an Option with the difference that when the last action of Loop is performed the condition of the loop shall be again checked. If it evaluates to true, the first action of the loop is performed (message 2). Otherwise, the first action after Loop is performed (message 4). Therefore, both the first action of Loop and the first action after Loop are added as performable actions. Fig. 6(d) illustrates the CTMC for the SD with Loop (Fig. 2(d)). Alternative fragment is also treated similarly to Option. The difference is that when the first action of the first operand (message 2) is extracted, the first actions of the other operands (message 3) are added as performable actions (Fig. 6(c) shows the output CTMC). Unlike the other combined fragments, the

Parallel structure is simply transformed by using interleaving techniques. In fact, when the first action of the first operand is extracted, the first actions of the other operands are extracted and added as performable actions (Fig. 6(e)).



(a) Basic (b) Option (c) Alternative (d) Loop (e) Parallel

Fig. 6 The resultant CTMCs for the SDs shown in Fig. 2

3.3 Transforming SDs into Reward DTMCs

An SD is transformed into a Reward DTMC to verify cost properties specified in cost/reward logic. This transformation works similarly to the transformation we described for DTMC except that a cost value is attached to each transition that corresponds to a message.

4 The Framework at Work

Hereafter we describe how our approach can be applied on a running example of a health-care service provided for people suffering from life-threatening illnesses. Because of their physical problems, patients should be in touch with medical centers, and be able to get continuous assistance. To use the service, each person is required to carry a small device which includes some sensors and is capable to monitor and store vital data. Whenever a person needs a medical help, he or she pushes the Help button on the device. The device automatically sends the vital data and the position of the person to the health care center. To detect the position, the device finds and uses one of available positioning services provided by an external device (GPS or GSM), and searches for a communication network to transmit data to the center. After the center receives a help request, it requests the laboratory for medical analysis and problem diagnosis. Based on the result and the severity of the problem, the center decides how to help the person. In the end, the center performs two actions concurrently: It informs the emergency team and also notifies the patient. Fig. 7 shows the SD diagram of the above scenario. The

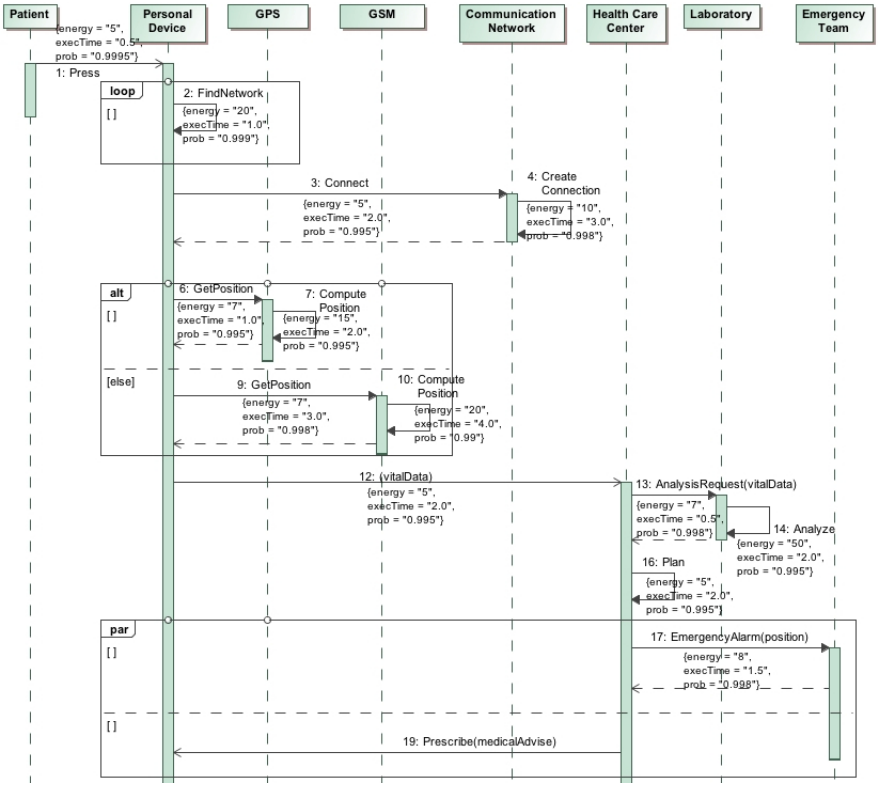


Fig. 7 The SD for the case study[1]

diagram is annotated with reliability, performance, and cost assumptions. As for cost, we consider energy consumption by the application.

Let us consider the following informal performance requirement for the running example: the emergency team shall be informed within 40 seconds after the button is pushed. The requirement can be rephrased in Structured English using ProProST as follows:

The system shall have a behavior where with a probability greater or equal 0.90 it is the case that (EmergencyTeam = Informed) will eventually hold within 40 seconds. From this property formulation in Structured English, we can automatically derive corresponding CSL property:

$$P \geq 0.90\left[(true) \cup \leq 40(state = 18 \mid state = 23)\right] \tag{1}$$

[1] For readability reasons we did not include in the annotations the probabilities associated with Loop and Alternative. However, we assume the probability of iterating is 0.66 and the probability of THEN branch of Alternative is 0.7.

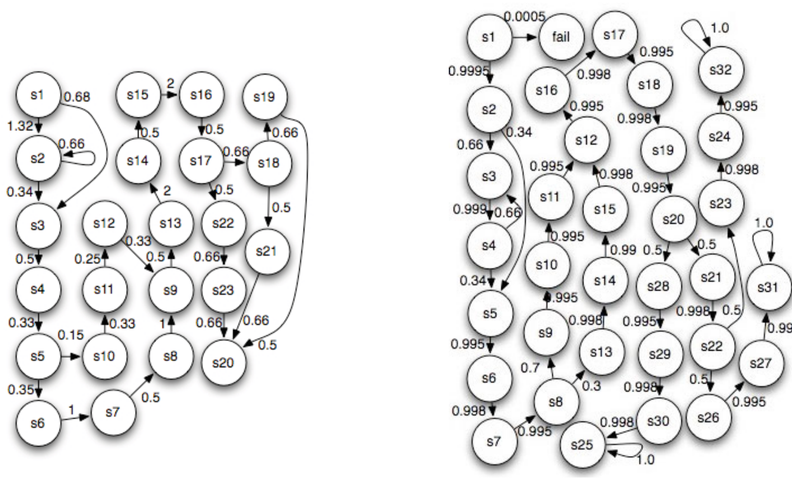
This CSL formula is verified against the CTMC (Fig. 8(a)), which is derived from the SD in Fig. 7. Using the PRISM model checker, we can verify the property. Actually PRISM tells us that the estimated probability for this property is 0.96, so the performance requirement is successfully satisfied. Regarding the CSL formula, note that the states 18 and 23 are the states exactly after the occurrence of the message 17, when the emergency team is informed. The reason that there are two states in the CSL formula is due to the interleaving of the messages 17 and 19. In other words, the message 17 can be transmitted either before or after the message 19. Let us now consider the following reliability requirement, expressed in ProProST:

The system shall have a behavior where with a probability greater or equal 0.97 it is the case that ((Person = Notified and HealthCenter = InformedByTeam) will eventually hold. The corresponding PCTL formula is:

$$P \geq 0.97\left[F\left(state = 25 \mid state = 31 \mid state = 32\right)\right] \quad (2)$$

The PCTL formula is verified against the DTMC in Fig. 8(b), which is derived from the SD in Fig. 7. Note that because of the complexity of the DTMC, we did not include failure transitions and states. The only failure shown is for message 1. Given the DTMC, PRISM tells us that the property is evaluated with probability 0.94. The requirement says that the probability must be above 0.97, so it is violated. Note that states 25, 31, and 32 are the states exactly after having the messages 18 and 19 transmitted.

Let us now consider a requirement expressed as a cost property about energy. According to the requirement, the total amount of the energy needed for the scenario shall be less or equal 200. This may be expressed in our extension of ProProST as:



(a) CTMC (b) DTMC

Fig. 8 The output of the transformation

The system shall have a behavior where with a cost less or equal 200 it is the case that (Person = Notified and HealthCenter = InformedByTeam) will eventually hold. In turn, this is translated into cost/reward logic as:

$$R \leq 200\left[F\left(state = 25 \mid state = 31 \mid state = 32\right)\right] \tag{3}$$

Note that the states 25, 31 and 32 are the states in which the patient is notified after receiving the message 19 from Health Center and also Health Center has received the message 18 from Emergency Team. The result of verification is about 180, which properly satisfies the property.

5 Related Work

Several approaches exist to transform UML diagrams into other formalism and then take advantage of existing tools for such formalism. For example, in [17] and [11] UML diagrams are mapped onto stochastic Petri Nets and then tools like SPE [11] or TimeNET [14] are used for performance evaluation. In [9], the authors discuss performance analysis of an activity diagram decorated with time constraints through probabilistic model checking. In fact, they transform a diagram into a DTMC (in terms of Prism language) and then verify their discrete-time properties. To perform performance analysis, [15-16] transform UML activity and sequence diagrams into the PEPA performance language. In [1], the authors give general guidelines to make a Markov chain from a given SD diagram. However, they never discuss how to use the generated models for performance analysis. Cheung et al [5] propose a framework to assess reliability of a component-based architecture. The main idea of the framework is to specify state-based behavior of components and to generate a Markov chain to assess the reliability. In [6], the authors take SD diagrams and deployment diagrams, and formulate a reliability model using a mathematical equation, by which the reliability of the system is calculated. Moreover, some intermediate languages CSM [18], Klaper [7], and Palladio [4] have been proposed to provide the bridges between generic high-level models like UML and analyzable low-level models like Queuing Networks and Markov chains. They essentially propose meta-models, which abstract away irrelevant data with respect to performance and reliability. To best of our knowledge, our framework is the only work that starts with both system models and requirements and ends up with probabilistic models. We cover three major non-functional requirements (performance, reliability and cost) in the framework and provide logic and analyzable formalism and tools for all of them.

6 Conclusions and Future Work

In this paper, a model-based framework has been proposed by which non-functional requirements can be analyzed with respect to system models developed at an early stage of development. The core part of the framework includes

different model-to-model transformations from SDs to Markov models, each of which is used to analyze a particular non-functional requirement. U-MarMo is under implementation, and at this time the mapping from UML SDs to DTMC is complete. As future work, we plan to move the framework and use it for run-time adaptation. Also, we are considering non-determinism as another way to express uncertainty. To support non-determinism, we consider using Markov Decision Processes.

Acknowledgements. This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom.

References

[1] Abdullatif, A.A., Pooley, R.: A computer assisted state marking method for extracting performance models from design models. IJSSST (8), 36–46 (2008)
[2] Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Proceedings of the 10th International Conference on Concurrency Theory, pp. 114–129 (1999)
[3] Ardagna, D., Ghezzi, C., Mirandola, R.: Rethinking the use of models in software architecture. In: QoSA, pp. 1–27 (2008)
[4] Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. J. Syst. Softw. 82(1), 3–22 (2009)
[5] Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L.: Early prediction of software component reliability. In: ICSE, pp. 111–120 (2008)
[6] Cortellessa, V., Singh, H., Cukic, B.: Early reliability assessment of uml based software models. In: WOSP, pp. 302–309 (2002)
[7] Grassi, V., Mirandola, R., Randazzo, E., Sabetta, A.: Klaper: An intermediate language for model-driven predictive analysis of performance and reliability. In: CoCoME, pp. 327–356 (2007)
[8] Grunske, L.: pecification patterns for probabilistic quality properties. In: ICSE, pp. 31–40 (2008)
[9] Jarraya, Y., Soeanu, A., Debbabi, M., Hassaine, F.: Automatic verification and performance analysis of time-constrained sysml activity diagrams. In: ECBS, pp. 515–522 (2007)
[10] Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation, pp. 220–270 (2007)
[11] Merseguer, J., Campos, J.: Software performance modeling using uml and petri nets. In: MASCOTS Tutorials, pp. 265–289 (2003)
[12] MARTE Specification, `http://www.omgmarte.org/`
[13] PRISM Probabilistic Model Checker, `http://www.prismmodelchecker.org/`
[14] TimeNET Tool, `http://www.tu-ilmenau.de/fakia/8086.html/`
[15] Tribastone, M., Gilmore, S.: Automatic extraction of pepa performance models from uml activity diagrams annotated with the marte profile. In: WOSP, pp. 67–78 (2008)
[16] Tribastone, M., Gilmore, S.: Automatic translation of uml sequence diagrams into pepa models. In: QEST, pp. 205–214 (2008)

[17] Trowitzsch, J., Zimmermann, A., Hommel, G.: Towards quantitative analysis of real-time uml using stochastic petri nets. In: IODPS, pp. 139b (2005)
[18] Woodside, M., Petriu, D.C., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (puma). In: WOSP, pp. 1–12 (2005)
[19] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, pp. 411–420 (1999)
[20] UML 2.0, http://www.uml.org/

Testing Fault Susceptibility of a Satellite Power Controller

Marcin Iwiński[1] and Janusz Sosnowski[2]

[1] Institute of Computer Science, Warsaw University of Technology, ul. Nowowiejska 15/19, Warsaw 00-665, Poland
e-mail: M.Iwiński@ii.pw.edu.pl
[2] Institute of Computer Science, Warsaw University of Technology, ul. Nowowiejska 15/19, Warsaw 00-665, Poland
e-mail: J.Sosnowski@ii.pw.edu.pl

Abstract. The paper presents some experience with developing a satellite power controller. Due to high probability of transient faults (in particular caused by cosmic radiation) we had to check their impact on the controller operation. For this purpose we have developed a special test bed. It comprises some universal fault simulator (FITS), a software model of the controller and its environment. Simulation results show relatively high fault robustness. We also outline some further improvements of the controller code.

1 Introduction

In many applications microcontroller circuits with high dependability requirements are needed. Moreover they usually operate in harsh environment e.g. industrial, automotive, avionic, cosmic applications. Designing such systems we have to evaluate their fault susceptibility (dependability). For this purpose various fault injection techniques have been developed and described in the literature [1,2,4,5,7], including those developed in our Institute [7,8]. They provide a capability to simulate faults in the analysed system (or its model) and observe their effects in comparison with non –faulty behaviour (golden-run). Such experiments allow identification of critical points to mitigate fault effects with special techniques. This approach has been widely used in calculation oriented applications ([1,7,14] and references therein). Real time reactive applications (typical for microcontrollers) create more problems due to the need of considering system environment (e.g. the controlled object and its reaction to the controller signals) and more complex fault effect evaluation [4,10,12,21].

To deal with real-time applications we have developed an original fault injection methodology based on the fault injector FITS [6-9] enhanced with an interface to the model of the environment (control object) and special evaluation module which qualifies test results. These additional modules are application dependent however they use the provided interface to FITS. Hence an important issue is to gain more experience with various real-time systems, especially within

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 63–74.
springerlink.com

the above mentioned extensions of FITS. Some experiment results with industrial and automotive controllers have been described in [5,8,17-19]. This paper deals with a special satellite controller responsible for powering the satellite on-board electronics. The software of this controller has been adapted to PC environment in order to check its fault susceptibility with FITS simulator.

The prototype of the controller has been developed for a student satellite mission. It was implemented around COTS (commercial of the shelf) circuits (including Atmel microcontroller) . On–board equipment of satellites is exposed to cosmic radiation, which can result in single event upsets (SEUs) or latch-ups. These effects can disturb normal operation or even cause damages. The impact of cosmic radiation on electronic equipment has been studied in many publications (e.g. [3,11,16] and references). To reduce this impact we can use expensive radiation hardened circuitry or standard COTS elements supported with special fault effect mitigation techniques implemented in software. This approach is becoming more attractive in recent cosmic technology [3], however it needs deeper analysis of possible fault effects, for this purpose we can use fault injection techniques.

Section 2 descries the controller functionality, the developed testbed and experiment scenarios are presented in section 3. Experimental results based on simulation are illustrated in section 4 and concluded in section 5.

2 Satellite Power Controller

The main purpose of the satellite power supply unit (PSU) is to provide power to all other satellite subsystems. The electric power is produced by solar cells; during satellite flight the solar cells are susceptible to varying sun illumination which results in significant power fluctuation. To assure smooth powering the excess of the energy is stored in Li-Ion battery to maintain continuous operation during shadowed part of the orbit. To optimise the energy balance PSU uses a special control algorithm MPTT described in the sequel. Moreover the PSU performs some other supplementary functions:

- Providing overcurrent protection function for satellite subsystems (including latch-up elimination)
- Providing basic telecommands and telemetry
- Turning on and off other subsystems on demand (e.g. satellite payload)
- Collecting statistic data on various events or faults
- Autodiagnostics of PSU

The PSU software is organized as a control loop (with 10 ms period) performing MPPT calculations to deliver control signal PWM, monitoring currents and voltages, detecting latch-ups. Moreover it handles interrupts related to interfaces and watchdog timer. Events (compare fig. 1).

The PSU module has been implemented using ATmega1280 microcontroller. It is based on 8-bit AVR RISC architecture [22]. Microcontroller operates at 8 MHz clock and 3V power supply, which is best compromise between processing power and current consumption. It comprises CPU, on chip RAM (8kB), FLASH (128kB) some peripherals/(ADC converters) and interfaces used to monitor and control other functional blocks (shown in fig. 1). In particular this includes 3 pairs of solar cells (each pair relates to two parallel sides of the satellite cubic module), 3 DC/DC converters responsible for charging the battery from solar cells, 2 temperature sensors (digital thermometers with 1-wire interface - measuring the temperature of the battery and PSU circuit board), and 2 additional modules related to so called payload (PLD1 and PLD2). PSU uses UART and redundant SPI interfaces to exchange messages with satellite Communication Subsystem (COM) and its On-Board Computer (OBC). PSU has also outputs controlling P-MOSFET switches, which can be used to turn on or off all the subsystems, when latch-up occurs or appropriate command from Earth arrives through COM subsystem The DC/DC converters are controlled with PWM signals defining their duty cycles. Solar cells are connected in pairs separated by Shottky diodes and placed on opposite sides of the satellite. Hence only one solar cell within the pair is exposed to direct sunlight, so we need to use only 3 PWM channels and DC/DC converters instead of 6. All PSU inputs and outputs are shown on the edges of the graph in Fig 1. Most of microcontroller inputs are analogue inputs, which are responsible for measuring voltages and currents. PSU can also measure its own current using sense resistor and cut it off in case of latch up event. In this case the power restoration is delayed by about 500ms.

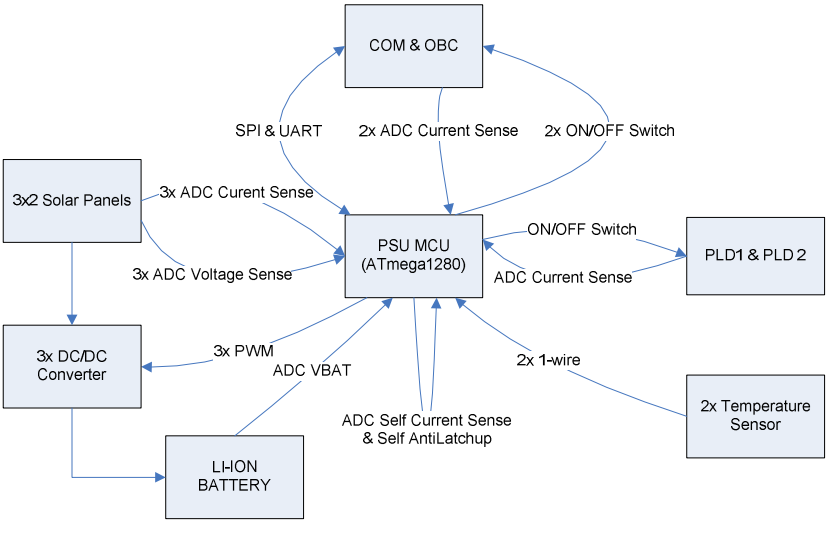


Fig. 1 PSU microcontroller inputs and outputs

Due to various light and temperature conditions solar cell current-voltage characteristics continuously change during satellite flight. Typical characteristics are shown on Fig. 2. Change of light intensity results in variation of maximum possible current which can be drawn from solar cell, while temperature influences maximum output voltage of the cell. The characteristic describing the solar cell output power in function of its voltage is also given in fig. 2. An important issue is to operate in the area of maximal output power.

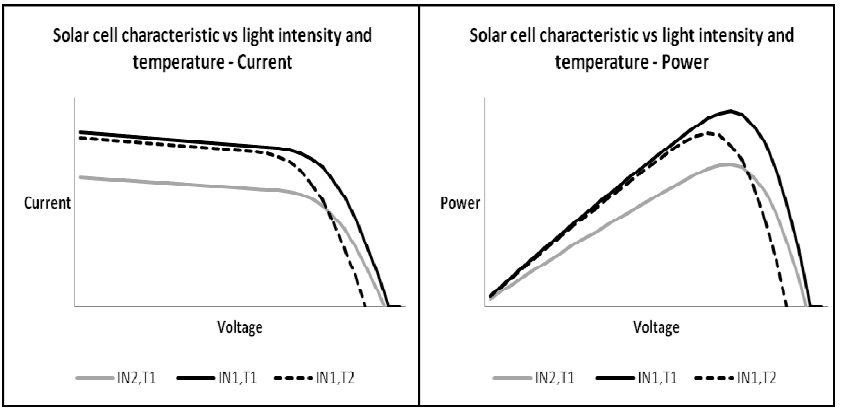


Fig. 2 Solar cell characteristics for two levels of light intensity IN1 > IN2 and temperature T1 < T2

To maintain maximum output power of the solar cells we have to use a special algorithm (MPPT - Maximum Power Point Tracking) to control DC/DC converters. This algorithm matches dynamic resistance on satellite power input with current solar panel characteristic to ensure that solar cell produce maximum available power. This is achieved with special DC-DC step-down switching regulators which present to the solar cells a load resistance dependent on controlling signal PWM. The microcontroller uses ADC channels to measure solar cells current and voltage, and then using implemented MPPT algorithm calculates PWM signal which determines the duty cycle for DC-DC converters.

Some MPPT algorithms are described in literature [13], and we have selected P&O (Perturb & Observe) version for the developed PSU. This algorithm provides good performance and uses only basic ALU operations: addition, subtraction and multiplication on integer values. More complicated MPPT-INC algorithm is 3% percent more accurate [15], but it requires integer division, which is only supported on 8-bit AVR architecture by software.

In the PSU the MPPT-P&O algorithm is executed every 10ms with new measured voltage and current values in appropriate points of the system. PWM control signal typically ranges from 0 to 80. This assures satisfactory control resolution and power tracking speed during satellite rotations (up to few rotations per minute).

3 Fault Injection Testbed

Fault injection experiments are based on FITS [8] injector, which has been developed in our Institute as a versatile fault injecting tool for Intel x86 platform. It provides an environment which works similarly to software debugger. FITS uses Windows Debugging API to control the execution of the application under tests (AUT). It can suspend/resume the AUT's threads, read/write AUT's memory (data, stack or code), CPU/FPU registers states, etc. FITS can emulate the faults of different types e.g. stuck-at or bit-flips by disturbing registers, memory cells etc. However, here we concentrate on single bit-flip faults which mimic Single Event Upsets (SEUs) [1,3,7].

Fault injection experiments usually involve many tests. Each test is a single execution of the AUT with the disturbance of a given type (injected fault). During the test FITS suspends the AUT at some time instant of its execution (fault triggering moment), injects the fault into a specified location (e.g. modifying CPU register, memory data or code area) and monitors fault effects after AUT resumes the execution. Fault effect analysis is based on the comparison of the registered AUT behaviour (generated exceptions, messages, timeouts, control signals, etc.) with the non-faulty reference run image (called golden run). In general, four classes of test results are distinguished: C (correct results produced), INC (incorrect/unacceptable results), S (test terminated by the system due to un-handled exception, e.g. memory access violation, invalid opcode), and T (timed-out test). Moreover, user defined messages (U) generated by AUT can also be included (e.g. related to detection of a specific error). Experiment configuration involves specification of the number of faults, their types, fault triggering and fault location scenarios. In most cases we use pseudorandom distribution of faults in time and space (location).

Checking fault susceptibility of reactive systems we have to provide not only the controlling application (AUT) but also its environment (controlled object). Here we present the results of testing only MPPT-P&O algorithm of the satellite controller. The controller environment is restricted to models of the solar cells and battery charging circuitry (DC/DC converters), they take into account changing external conditions (temperature, illumination, current load). Moreover defining test scenarios we have prepared various input data, which can represent changing environment parameters like light intensity, temperature or variable power load consistent with real operational conditions. AUT and its environment model have been integrated with FITS injector within a single IBM PC platform. However, faults can be injected only within the AUT. Executing each test FITS performs result qualification and provides us not only with individual test results but also with aggregated results for all tests within experiment (typically many hundreds of faults). To assure more accurate result qualification we have developed special module (coupled to FITS) in which we can define test result qualification conditions.

The performed experiments have been targeted at transient faults (bit flips), due to high probability of SEUs in the on-board electronic equipment of the orbiting satellite. We have considered two classes of transient faults: latched (bit flips in

memory cells, registers) and non-latched (temporary state changes). The non-latched transient faults simulate disturbances on bus, instruction register, and internal data paths. In particular they are especially interesting as program code disturbance, which can be related to controllers with programs stored in flash memories. State disturbance of this memory is of low probability, however temporary code disturbance (e.g. on the bus) during instruction fetching process is worth considering.

In the paper we present experiment results related to the most complex and important part of PSU controller i.e. MPPT-P&O algorithm. In particular we analyze the fault impact on the amount of power lost as compared with the golden run result. For this purpose we have developed a solar cell model described by the following equations:

$$\begin{cases} I = a\left(\frac{1}{40}U - \frac{35}{400} - \frac{b}{40} + 1\right) & U \le 3{,}5 + b \\ I = -a\left(U - 3{,}5 - b\right)^2 + a & U > 3{,}5 + b \end{cases} \tag{1}$$

$$I = U\left(\frac{1}{R} + \frac{4}{PWM}\right) \tag{2}$$

where: I - cell current [A], U - cell voltage[V], a - light intensity factor [0.0-1.0], b - temperature factor [-0.5,0.5], R - parallel resistance modeling solar cell load generated by PSU and other subsystems, PWM - PWM value generated by MPPT.

Equations 1 describe current dependence on voltage for given a,b values. The first part of the characteristic can be described as linear, while the second is generated by quadric equation. Equation 2 represents a dynamic resistance, which is the result of given PWM value. Solving Equation 1 with Equation 2 gives us voltage and current values defined by specified parameters a, b, R and PWM. For experimental simulations we have to provide a,b,R values for each iteration of the MPPT-P&O algorithm for each DC/DC channel. They relate to some considered scenario of temperature, illumination and load changes during satellite orbiting (operational environment). It is worth mentioning that parameter a (light intensity) can change relatively fast comparing to b (cell temperature). However R parameter remains usually constant for most of the experiment time changing only few times by a large step.

The tested algorithm has been implemented in version MPPT-P&O described in [13] and adapted to x86 platform with some small modification - optional restriction for minimal or maximal PWM value, to keep it in the allowed range. The algorithm uses as inputs appropriate voltage and current values delivered by the environment model and produces PWM signal as an output (controlling DC/DC converters - included in the environment model). The behavioral model of these converters is relatively simple: linear increase of input load (1/R) with PWM in the range [0, 80], for PWM > 80 the load visible by the solar cell pair is still

constant. The recent current and voltage signals provided by the environment model are stored in local variables used by the algorithm.

The block diagram of the test bed is given in fig. 3. The tested application (AUT) is coupled to the environment model which describes the behavior of solar cells (signals I(t), U(t)) in function of the PSU control signal PWM(t) and current load R(t) for every iteration t of the implemented algorithm (AUT). The experiment conditions are specified by the number of iterations (N) and tables comprising parameters a(t), b(t) and R(t) for each solar cell channel. FITS simulator injects faults into AUT according to the specified experiment configuration (number and types of injected faults, fault localization and distribution). Moreover, it collects results of each test (single fault injection) and produces the experiment statistics (over all performed tests). The qualification of the correct (C) and incorrect (INC) test results is performed by RQ module. This module monitors the behavior of environment (I(t), U(t)) and PSU (PWM(t)) in comparison with reference golden run parameters provided by golden run (GR) module. The test is considered as correct if the power (P(t) = I(t)*U(t)) deviates from the reference level less than a specified threshold - ΔP percent. To have a better insight into fault effects we can specify this condition as related to temporary power deviation (within one algorithm iteration) or as an average deviation within the whole test run (N iterations). It is also possible to register power characteristics in time for selected faults. This helps better fault interpretation.

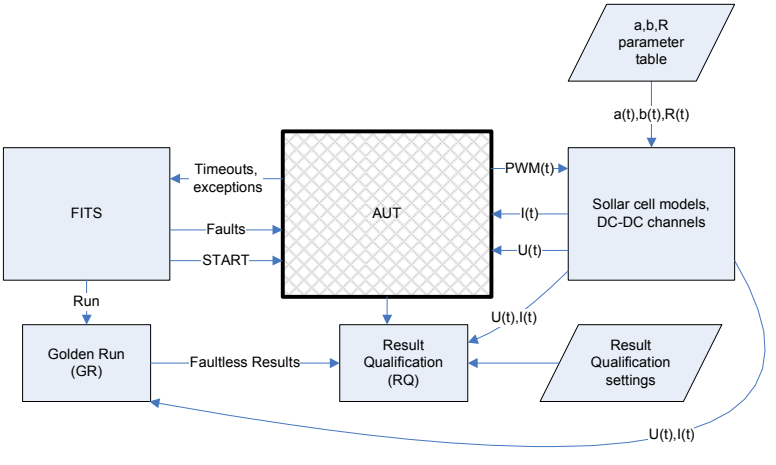


Fig. 3 Experiment testbed

4 Experimental Results

In the performed experiments we have injected bit flip faults in CPU registers, data and program code areas. Simulating faults in the program code two cases have been considered: latched (the program code is disturbed by a bit flip) and non-latched faults (the code is disturbed by a bit flip, but after executing the disturbed

code it is recovered to the correct state). The latched faults are typical for microcontrollers fetching program code from RAM or cache memories. The non-latched faults relate to the case of codes fetched directly from flash memory (the fault models here disturbances on the bus or instruction register - the state of flash is assumed to be robust).

In tab. 1 and 2 we give a sample of results of fault injections into the program code (latched and non-latched respectively). These results relate to 11 different configurations of experiments (the first column). For each experiment N=500 faults have been injected randomly (in time and space). The basic experiments deal with a single DC/DC channel and assume PWM signal bounding in the range [0, 80]. Experiments marked with asterix (*) take into account two independent DC-DC channels, which shared only algorithm code and static data. Experiments with id in underlined bold (6, 7) relate to the algorithm with no bounds on PWM signal. The column ΔP specifies the assumed maximal percentage of power losses to qualify the tests as correct. Here we also distinguish result qualification based on temporary or averaged (underlined bold) power losses (compare section 3).

Table 1 Experimental results for non-latched faults (in program code)

Experiment number	ΔP [%]	Result correct (%)	Result incorrect (%)	Terminated by OS (%)
1	10%	47,80	9,20	43,00
2	30%	44,40	9,80	45,80
3	**0,1%**	42,20	10,80	47,00
4	**1%**	45,20	8,60	46,20
5	**5%**	53,80	0,40	45,80
6	**1%**	48,80	0,60	50,60
7	**0,1%**	46,00	5,40	48,60
8*	10%	45,40	11,00	43,60
9*	**1%**	50,40	4,60	45,00
10*	**5%**	55,40	0,20	44,40
11*	**1%**	56,80	2,20	41,00

All experiments covered 500 iterations of the control algorithm. The environment has been defined by a table comprising 500 rows with specified a, b and R parameters (for each iteration). Experiments with 2 DC/DC channels needed two sets of such tables. Parameter a (related to light intensity) may change significantly during satellite mission, so we model a few changes of this parameter during the whole simulation time (500 iterations) from an almost minimal to almost maximal values. Parameter b (related to temperature factor) changes its value slowly due to quite large thermal capacity of the satellite. So we simulate slow changes in the range of 10% of the initial value. Defining parameter R we have assumed one additional load (e.g. related to switching in the communication module) for 50 iterations in the middle of the simulation time.

Table 2 Experimental results for latched faults (in program code).

Experiment number	ΔP [%]	Result correct (%)	Result incorrect (%)	Terminated by OS (%)
1	10%	18,40	38,80	42,80
2	30%	21,80	30,60	47,60
3	**0,1%**	14,80	37,80	47,40
4	**1%**	16,20	40,00	43,40
5	**5%**	24,40	24,00	51,00
6	**1%**	22,40	29,20	48,40
7	**0,1%**	16,80	32,60	50,60
8*	10%	11,60	39,00	48,40
9*	**1%**	18,20	35,40	45,20
10*	**5%**	26,40	28,00	45,20
11*	**1%**	29,80	23,80	45,20

Comparing results of tab. 1 and 2 we observe much lower percentage of incorrect results for non-latched faults than for latched ones. This confirms that flash memory used for the program code improved system robustness significantly as compare with RAM or cache. Injecting bit flips into CPU registers we observed typically 1-4.2% of incorrect results and 36-43% system exceptions. Faults in data area resulted in 0-2.7% of incorrect results and no exceptions (so 97-100% results were correct). Timeouts contributed less than 1.6% in the case of faults in code, for other fault locations it was practically close to 0. Relatively high fault robustness for register and data memory cell faults results from the iterative nature of the algorithm and some kind of self-repair in calculations. Faults resulting in exceptions can be easily handled by starting the algorithm in a predefined state. It is worth noting that increasing the acceptable power loss level (ΔP) we get higher percentage of correct results.

The presented test platform provides also possibility to record PWM value behaviour for every single test. From thousands of test runs we have chosen two interesting simulation plots. Figure 4 represents PWM value during one of test runs in function of the iteration number (equivalent to time). Black line represents PWM value during fault injection run, while the grey one is faultless run. Most of the time black line covers the grey one. In iteration 141 a non-latched fault in the instruction code has been injected. The fault caused corruption of a compare instruction responsible for keeping PWM value in desired range. PWM value was changed to zero due to faulty behaviour of this mechanism. In the subsequent iterations MPPT-P&O algorithm tried to recover from this state and finally it succeeded. The plot representing the provided power is similar, which means that a few percents of possible power were lost due to this fault.

Figure 5 presents result of latched fault injection into instruction code. Contrasting to figure 4, this time PWM value continued to increase as a result of fault. The MPPT-P&O algorithm could not recover itself from this state causing significant power loss starting at the time of fault injection (iteration 379). We investigated what exactly happened resulting in so unpredictable behaviour of the tested

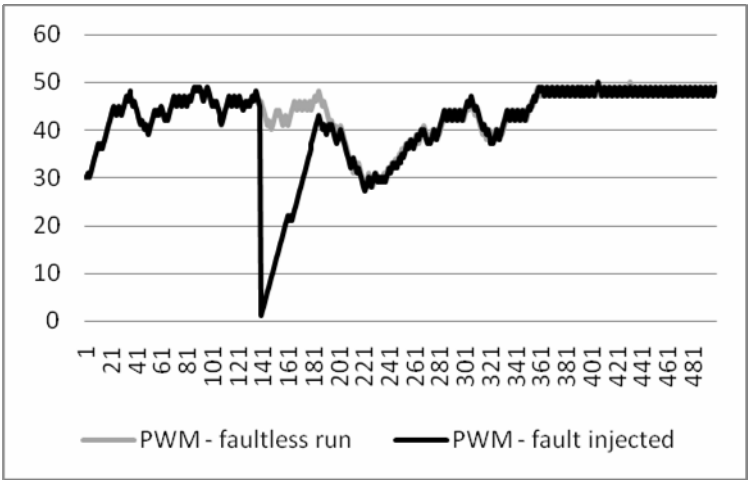


Fig. 4 PWM disturbance with a non-latched fault

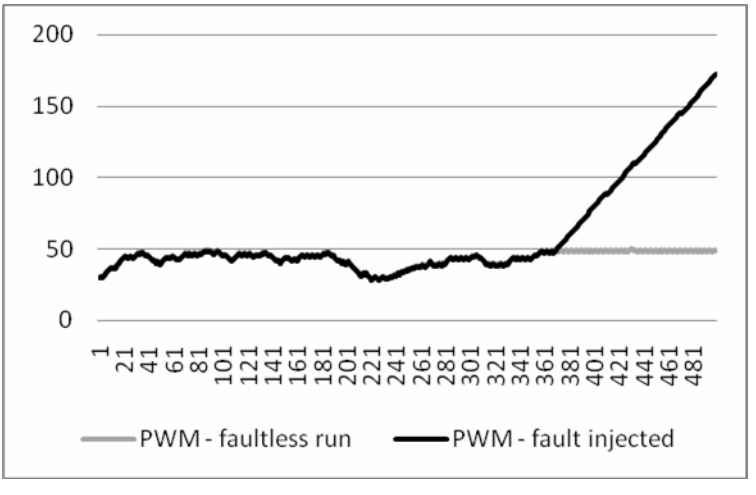


Fig. 5 PWM disturbance with a latched fault

algorithm. The reason was a disturbance of a compare instruction, causing its condition to be always true. This resulted in steady PWM value increase in following iterations.

Fault robustness can be improved with various software mechanisms e.g. based on control flow checking, data redundancy, reinitilization of variables or operational modes of interfaces, etc. [3,7,9,12,14,17-20]. Some of these approaches have been also incorporated in the PSU, as well as some autodiagnostic programs.

5 Conclusion

The paper confirms that FITS simulator together with the controller and its environment models assure high experiment controllability and observability Moreover, the application dependent test qualification is an efficient approach to fault injection experiments with reactive systems. An important issue is to assure some controllability in defining different classes of the system behavior (in our case the level of power anomalies). Basing on FITS fault injector (targeted at Windows environment and Intelx86 platform) we could only deal with an appropriate model of the controller. Nevertheless it allows us to identify the algorithm sensitivity to faults and to find the most critical points for improvement (the data and control flow of the controller and its model are compatible). Recently we develop a dedicated fault injector targeted at the Atmel platform, which will allow us to perform the experiments with the real physical controller. Moreover, further research will deal with fault detection and fault handling mechanisms (including autodiagnostics).

As compared with our previous experience with reactive system e.g. [6,18-20] in the satellite controller we also observe quite high natural fault robustness higher for higher level of accepted power deviation. Moreover, simple software mechanisms are sufficient to assure relatively high dependability especially for the controller with flash program memory.

We express our appreciation to P. Gawkowski for consulting the usage of FITS simulator in the experiments.

References

[1] Arlat, et al.: Comparison of physical and software-implemented fault injection techniques. IEEE Trans. on Computers 52(9), 1115–1133 (2003)

[2] Benso, A., Prinetto, P.: Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation. Kluwer Academic Publishers, Boston (2003)

[3] Campagna, S., Violante, M.: A framework to support the design of COTS based reliable space computers for on-board data handling. In: Proc. of IEEE IOLTS Symposium, pp. 91–96 (2010)

[4] Cunha, J.C., et al.: A study of failure models in feedback control systems. In: Proc. International Conference on Dependable Systems and Networks DSN 2001, Goteborg, Sweden, pp. 314–326 (2001)

[5] Fidalgo, A.V., et al.: Real Time Fault Injection Using a Modified Debugging Infrastructure. In: Proceedings of the 12th IEEE International Symposium on On-Line Testing (2006)

[6] Gawkowski, P., et al.: Software implementation of explicit DMC algorithm with improved dependability. In: Novel Algorithms and Techniques in Telecommunications Automation and Industrial Electronics, pp. 214–219. Springer, Heidelberg (2008)

[7] Gawkowski, P., Sosnowski, J.: Dependability evaluation with fault injection experiments. IEICE Transactions on Information & System E86-D, 2642–2649 (2003)

[8] Gawkowski, P., Sosnowski, J.: Experiences with software implemented fault injection. In: Proc. of the International Conference on Architecture of Computing Systems, pp. 73–80. VDE Verlag, GMBH (2007)

[9] Gawkowski, P., Grochowski, K., Ławryńczuk, M., Marusak, P., Sosnowski, J., Tatjewski, P.: Testing Fault Robustness of Model Predictive Control Algorithms. In: Giese, H. (ed.) ISARCS 2010. LNCS, vol. 6150, pp. 109–124. Springer, Heidelberg (2010)

[10] Kopetz, H.: Real-Time Systems - Design Principles for Distributed Embedded Applications. Kluwer Academic, Netherlands (1998)

[11] Lovelette, M.N., et al.: Strategies for Fault-Tolerant, Space-Based Computing: Lessons Learned from the ARGOS Testbed. In: Aerospace Conference Proceedings, vol. 5, pp. 5-2109–5-2119 (2002)

[12] Muranho, J., et al.: Failure boundness in discrete applications. In: Proc. 3rd Latin-American Symposium on Dependable Computing, Morella, Mexico, pp. 160–169 (2007)

[13] Omole, A.: Analysis, Modeling and Simulation of Optimal Power Tracking of Multiple-Modules of Paralleled Solar Cell Systems. The Florida State University/College of Engineering (2006)

[14] Rebaudengo, M., Reorda, M., Villante, M.: A new software based technique for low cost fault tolerant application. In: Proc. of IEEE Annual Reliability and Maintainability Symposium, pp. 23–28 (2003)

[15] Sera, D., Kerekes, T., Teodorescu, R., Blaabjerg, F.: Improved MPPT algorithms for rapidly changing environmental conditions. In: 12th International Power Electronics and Motion Control Conference, pp. 1614–1616 (2006)

[16] Shirvani, P.P., McCluskey, E.J.: Fault-Tolerant Systems in a Space Environment: The CRC ARGOS Project, CRCTR 98-2. Stanford University, Stanford (December 1998)

[17] Skarin, D., Karlsson, J.: Software implemented detection and recovery of soft errors in a break by wire system. In: Proc. of 7th European Dependable Computing Conference, pp. 145–154. IEEE Comp. Soc, Los Alamitos (2008)

[18] Trawczynski, D., Sosnowski, J., Gawkowski, P.: Analyzing Fault Susceptibility of ABS Microcontroller. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 360–372. Springer, Heidelberg (2008)

[19] Trawczynski, D., Sosnowski, J., Gawkowski, P.: Testing Distributed ABS System with Fault Injection. In: Proc. International Joint On-Line Conference on Computer, Information, and System Sciences, and Engineering – CISSE 2009, On-line Conf. (2009)

[20] Trawczynski, D., Sosnowski, J.: Delayed based SWIFI approach to ABS dependability. In: Sugier., J., et al. (eds.) Technical Approach to Dependability, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2010, Poland, pp. 147–158 (2010), ISBN 978-83-7493-528-9

[21] Vinter, J., et al.: Experimental dependability evaluation of a fail-bounded jet engine control system for unmanned aerial vehicles. In: Proc. International Conference on Dependable Systems and Networks DSN 2005, Yokohama, Japan, pp. 666–671 (2005)

[22] ATmega640/1280/1281/2560/2561 Preliminary, revision L, ATMEL Corporation (2007), (updated September 2007)

Theoretical and Practical Aspects of Encrypted Containers Detection - Digital Forensics Approach

Ireneusz Jozwiak[1], Michal Kedziora[2], and Aleksandra Melinska[3]

[1] Wroclaw University of Technology, Wroclaw, Poland
[2] Wroclaw University of Technology, Wroclaw
email: Michal.kedziora@pwr.wroc.pl
[3] Wroclaw University of Technology, Wroclaw, Poland

Abstract. This paper covers problem of detecting encrypted files in evidence data during digital forensics investigations. We present comparison of popular detection methods like file signature and extension analysis, metadata analysis and searching operation system artifacts. We present research on theoretical and practical use of some indicators that can suggest encryption used like entropy, chi-square test, Arithmetic Mean and Monte Carlo Value for Pi.

1 Introduction

This paper describes theoretical and practical aspects of detecting encrypted data and files during computer forensics investigation. We start with simple and obvious methods like file signatures analysis, but our main research is done with advanced mathematical statistical methods of encryption detection that can be used nowadays.

Digital forensics investigation is wide and complicated process. One of popular definition says that computer forensic is "the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitation or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations"[17], so we can see clearly that one of the main objective of computer forensic investigation is to search and analyze any data and files which may contain any interesting information.

Naturally one of the first steps, which people do to make their information safe is to hide it. It is not so unusual that users move valuable files into system folders or rename them to make them inconspicuous[6]. Of course usually this methods are not much effective, further more in the most cases those files are highlighted and revealed on the beginning of investigation. Some of techniques used for that aim will be described later e.g. hash analysis, binary search, time line analysis and signature analysis. More advanced users are aware that hiding data in the best case will delay revealing of evidences, but in most cases it will make it even faster[7].

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 75 – 85.
springerlink.com

That is one of the reason why users encrypt valuable data[13]. It should be said here clear that encryption is not reserved for criminals who want to hide evidences of illegal actions. Encryption is part of computer security, we can encrypt data to make it safe from being steeled, from competition, sometimes legal regulations force us to encrypt fragile data. Whatever reason of encryption is, the first question digital forensic investigator will ask himself will be why suspect encrypts his data, maybe he want to hide something important. That is why it should be priority to find those files in computer forensic investigation process.

There is plenty of free and commercial tools to encrypt and decrypt data[19] and to create encrypted containers where users can copy files to make them encrypted. We decided not to describe different cryptographic algorithms because from practice we can observe that it does not matter. It is because in the most computer forensic cases there is no need to perform full cryptanalysis process. It is because most of cryptographic algorithms are based on Auguste Kerckhoffs rule that says that algorithms security must be based on well-known algorithm and secret password. For sure this rule is positive, but there is a huge sore point of this approach – password. Research are pointing that most users use very week passwords[20]. Microsoft research which involved half a million users over a three month period have given result of average password bit strength equal 37.86. This means passwords where short, included mainly lowercase letters, without uppercase, digits and special characters. Other research based on password lists revealed by hackers can conclude that most of password are short words which can be find in dictionary, or words related to personal details of user (name of pet, date of birth). For sure it is huge issue for security professionals how to convince users to use harder passwords, but for forensic investigators it is easy ability to guess password and analyze decrypted data. Next problem with passwords (or more with users) is that one password is almost for sure used in several applications (mail, webpage, logins). This means that during investigations we can find hardly protected passwords from e.g. mail or internet browser (where they are typically stored in clear text), and then use this passwords to decrypt files. Third way to get encrypted data password during digital forensic investigation is to ask suspect (or ask prosecutor to do it)[9][10]. In some law systems for example in UK, if you will not give passwords that authority believe you possess, you will have to serve a sentence 5 years in prison for failing to comply with police or military orders to hand over either the cryptographic keys, or the data in a decrypted form. Actually 5 years imprisonment is reserved for terrorism cases, and 2 years sentence in any other cases[8].

When we put all this together we can conclude that in contemporary computer forensic investigations, the biggest problem is not to decrypt encrypted files but to find those encrypted. The chapter is divided into three parts. The first part will present methods of detecting encrypted files based on file metadata. These are simple and fast methods to discover encrypted files. The second part will present advanced statistical algorithms we can use to detect not only encrypted files but also encrypted data hidden in unallocated space. In the third part we present

indirect methods to detect that encrypted files can be found on the disk. Those methods do not point to encrypted files but they tell that is high possibility that those files are located on a disk.

2 Signature Analysis

Methods presented in this section are based not on encrypted data inside a file, but on file metadata. This methods are not quite accurate but there are fast and in some cases they can give immediate results.

First method is based on searching files with specific extensions[11]. Encryption tools usually create unique filename extensions to identify them by operation system. This process is called Application Binding and it is very comfortable situation for user because operation system can associate encrypted file with specific software and decrypt it with minimal user action. From computer forensic point of view searching files by specific extension is the easiest way to find specific encrypted files. Some example of encrypted files extensions: .aes (AES Crypt encrypted file), .asc (Pretty Good Privacy armored encrypted file), .axx (Encrypted file), .cef (SanDisk CruzerLock encrypted file), .cry (Cryptainer encrypted volume file format), .dez (DES encrypted zip file), .drc (DriveCrypt container file), .fcfe (Microsoft Access encrypted database), .jbc (BestCrypt file), .pgp (Pretty Good Privacy encrypted file), .sde (Steganos Disk Encryption file format). We must know that Windows Operation Systems uses extensions to associate files with applications but in Unix not extension but header is mostly used. In MAC OS is used combination of extension, header and 32bit metadata information called file type code. Process of identifying and comparing extensions, headers and footers of files is called file signature analysis. It is part of standard digital forensic investigation procedure that is why it can be successfully used also to detect files which signatures point to be encrypted files. The signature analysis can also detect files which extensions were changed on purpose, to mislead investigation (it is common to change .jpg or .zip files into .sys, or .temp). Performing signature analysis we can have four outputs[5]. First is when header of file is known and extension is also known and mach. This is absolutely normal situation where file e.g. example.zip as Zlock pro encrypted ZIP has file header equal 50 4B 03 04 14 00 01 00 63 00 00 00 00 00 and extension ZIP. Second situation is when header and extension is known, but extension not match with header, corresponding to last example it would be example.dll, where someone intentionally has changed extension to mislead investigation. For sure this attempt will be discovered in a short time. Third situation is when file header and extension are both unknown, this situation can occur by dealing with e.g temp files. Fourth situation is when header is unknown and extension is known but doesn't match. This case can be spot during processing True Crypt files. Encrypted True Crypt containers usually have default .tc extension but it is not mandatory. True crypt containers can have any extension to work properly and file itself does not have any signature. Often true crypt encrypted containers have extensions of other files to mislead investigation. It can also be used as a trace because relatively small amount of files are without any

header. As we can see signature analysis is a powerful tool to discover encrypted files in operation system, but still we have to realize that this method does not detect encryption used but points that file was created by cryptographic tool. To confirm that file has encrypted data inside we have to adopt statistical algorithms.

3 Statistical Data Analysis Algorithms

To detect encrypted data in the most accurate way, we have to understand mathematical fundaments of cryptography. As a definition encryption function εk with key k, where c is cipher text, p is plaintext is equal to:

$$\mathrm{c} = \varepsilon_{\mathrm{k}}(\mathrm{p}) \tag{1}$$

In the beginning of cryptography history there where simple substitution ciphers which weren't very strong. Cryptanalyst could easily guess massage by analysing the frequency distribution of the cipher text. In the course of time ciphers were evolving into block ciphers, which were not so easily broken with frequency analysis. One of the techniques to prevent cryptanalysis of cipher, was called whitening[15]. At the end cipher text started to look like random data stream. It helped to make cryptographic algorithms stronger, but it also made possible to identify encrypted data by measuring randomness of data[4]. In fact there is no other simple explanation to keep random data then the file is encrypted massage. Statistical data analysis encryption detection algorithms are based on statement that encrypted massage is similar to pseudorandom data[14]. There are several methods and techniques to test data sequences to be random. Most of them are based on Golomb's randomness postulates[12]. The first postulate tells that in the cycle N of s, the number of 1 differs from the number of 0 by at most 1. Second postulate tells that in the cycle N, at least half the runs have length 1, at least one fourth have length 2, at least one eighth have length 3, etc., as long as the number of runs so indicated exceeds 1. Moreover, for each of these lengths, there should be equally many gaps and blocks. Third postulate tells that the autocorrelation function C(t) is two valued. That is for some integer K,

$$N \cdot C(t) = \sum_{i=0}^{n-1} (2s_i - 1) \cdot (2s_{i+1} - 1) = \begin{cases} N, & t = 0 \\ K, & 1 \le t \le N-1 \end{cases} \tag{2}$$

These postulates are not sufficient to consider data as random but there are necessary to be fulfilled by all random data. Next we present chosen algorithms which can be used to determine if data is much or less random.

3.1 Entropy Based Detection

First factor we present will be entropy. Originally entropy comes from physics and thermodynamics and is a measure of the disorder or randomness of the

constituents of a thermodynamic system. Entropy was adopted into computer science where it represents measure of the uncertainty associated with a random variable[1]. From definition entropy H of a discrete random variable X with possible values $\{x_1,\ldots,x_n\}$ is equal:

$$H(X) = E(I(X)) \tag{3}$$

Where I is the information content of X. I(X) is a random variable and E is the expected value. If p denotes the probability mass function of X then entropy is equal to:

$$H(X) = \sum_{i=1}^{n} p(x_i)I(x_i) == -\sum_{i=1}^{n} p(x_i)\log_b p(x_i), \tag{4}$$

Where b is the base of the logarithm. Entropy value will be close to max value when the input will be random data. Any signs of data order will lower entropy value. We can predict efficiency issues while using this algorithm to compute entropy, it is because we have to compute logarithms. To avoid this some encryption detection tools are implemented with simple other simpler randomness tests[3].

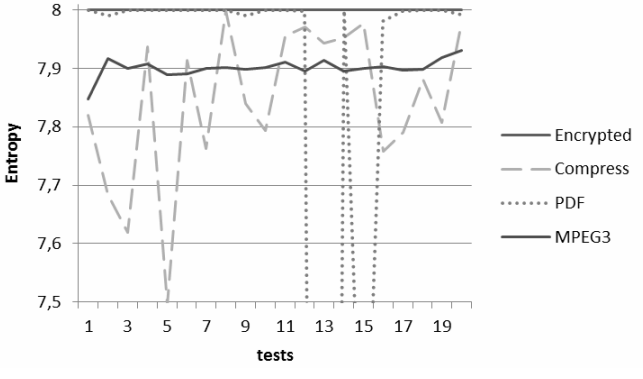


Fig. 1 Entropy test values

After testing various types of files we received range of encrypted data to be from 7,99984 to 7.99999. Compressed files (mainly we used files with zip and rar compression algorithms) where just slightly lower, but difference is almost unnoticeable. This tells as that method of encrypted data detection can have false positive hits with some compressed files. With other types of files we do not have such doubts. PDF files have entropy based of kind of data they consist, but in most cases entropy level is much lower than 7,99. MPEG3 files have entropy at level around 7,9 and its constant in different files, it can be explained by compression algorithm used in this file types.

3.2 Chi Square Test

Chi Square test is a statistical hypothesis test in which the distribution of the test is a chi-square distribution when the null hypothesis is true[2]. There are several tests build on this assumption, but the main use is to confirm randomness of data. From the definition chi-square[16] test with k probable outcomes, performed n times, in which Y_1, Y_2, Y_3… Y_k is the number of experiments which resulted in outcome, where the probabilities of each outcome are p_1, p_2,… p_k is:

$$\chi^2 = \sum_{1<s<k} \frac{(y_s - np_s)2}{np_s} \tag{5}$$

In result we should expect the lower chi square sum for more random data. From a chi-square, the probability Q that the X^2 sum for the test with d degrees of freedom is regular with null hypothesis and can be compute as:

$$Qx^2, \mathrm{d} = \left[2^{\mathrm{d}/2} \Gamma(\frac{d}{2}) \right]^{-1} \int_{x^2}^{\infty} (t)^{\frac{d}{2}-1} e^{-\frac{t}{2}} dt \tag{6}$$

Where Γ is a factorial function to complex and real arguments:

$$\Gamma_x = \int_0^{\infty} t^{x-1} e^{-t} dt \tag{7}$$

We have performed several tests with different encrypted, compressed, MPEG and PDF files. Encrypted files had chi square on constant value of near to 256.Any other files had Chi Square value thousands or millions higher. For compressed files average value was 16276778 It is 63581 times higher than Chi Square value of encrypted files. For MPEG and PDF files values where successively 9157435 and 5131275. Chi Square is extremely sensitive, that is the reason why it is used to check pseudorandom generators. It is also accurate way to detect encrypted files and distinguish them from any other.

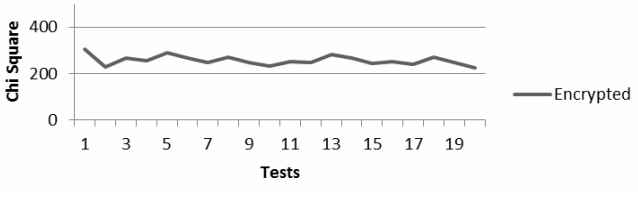


Fig. 2 Chi Square test values

3.3 Other Algorithms

There are other random detection tests like Frequency tests. For example monobit test which checks if number of values in data stream is near equal, for two bit we have:

$$X = \frac{(n_0 - n_1)^2}{n} \tag{8}$$

For bytes when we get 255 possibilities we can use modification of frequency test which is ease to implement based on equation:

$$X_h = n * \frac{\min(hist)}{\max(hist)} \tag{9}$$

This method of calculating entropy is based on histogram of data we are interested with. We take minimum value from histogram and divide it to maximum value. The smaller difference between these two values, there is the more random data. Output of division will give entropy near one for random data, and less random data will have near zero value. Output is multiplied by 8 to make it compatible with definition of entropy. To make output more comparable to entropy calculated from definition, outcome is multiplied by 8. Third simple of implementation of frequency test is summing all the bytes and divide it by the file length in bytes. We should get value about 127.5 for byte stream or 0.5 for bit stream. Any other value mean that data is not random from Golomb's randomness postulates.

We have performed a series of experiments on encrypted, compressed and other files. Result was that encrypted data is in conformity with random data frequency test and value is 127,5 ± 0,5. Compressed values are divergent but rarely in detection threshold of encrypted files. MPEG files have mean values much below encryption mean values in range between 124 and 125.

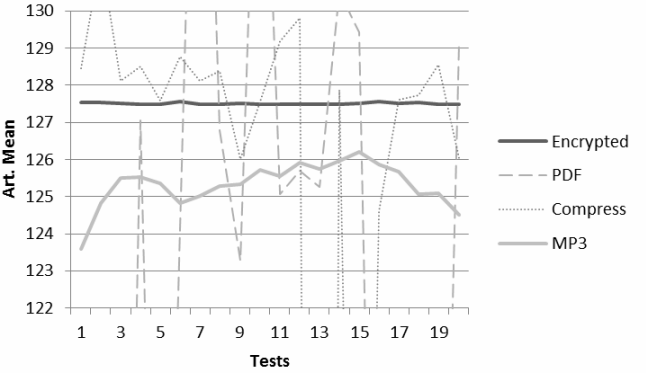


Fig. 3 Frequency Mean test values

Next algorithm is serial correlation X_{ac} which idea is to examine the correlations between the shifted sequences and it is computed from equation:

$$x_{ac} = \frac{2(A(d) - \frac{n - d}{2})}{\sqrt{n - d}} \tag{10}$$

Where $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$, d is fixed integer, and s is a tested sequence.

Monte Carlo Value for Pi is algorithm in which each following 6 bytes sequence is use as 24 bit X and Y coordinate. If the distance from random point is less than the radius of a circle placed in the square, sequence is called hit and the number of hits can be used to calculate the value of Pi. If the sequence is random, value should be equal to Pi value 3,14159265.

After performing series of tests we can conclude that Monte Carlo Pi algorithm is efficient in detecting encrypted data. All encrypted files are values near Pi value of 3,14. Compressed files have clearly lower (or higher) values, MPEG and PDF files are much outside encrypted file values.

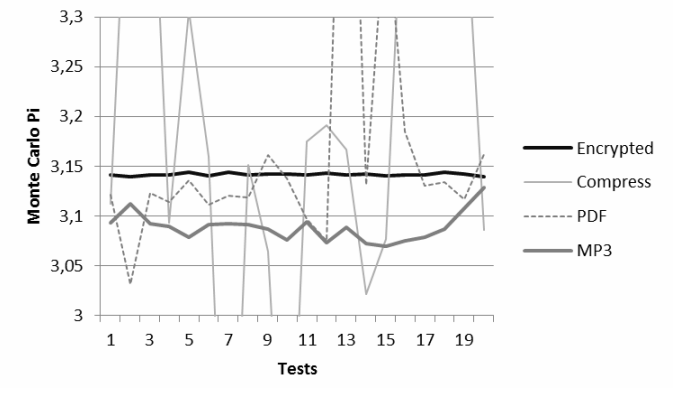


Fig. 4 Monte Carlo Pi test values

Other test algorithm which can be used for detection of random data are poker test which is generalization of frequency test for sequence values, two bit test which check occurrences of 00, 11, 01, 10 subsequence's, or runs test which checks number of chosen data blocks inside data sequence.

3.4 Data Length

One of the problem correlated with encrypted data detection based on statistical algorithms is length of input. If the data input will be to short algorithms will not have enough data to give accurate result. Main cryptographic tool used nowadays is True Crypt, fortunately minimum size of NTFS formatted encrypted container is 3792 KB. We prepared series of tests to check how file length corresponds with algorithms accuracy. We have performed described algorithms on several files from 148576 KB to 524288000 KB. Chosen results are presented on Table 1.

Table 1 Detection values with different file lengths

File Length	Entropy	Chi Square	Art. Mean	Monte Carlo Pi
1048576	7,99984	231,9458	127,4971	3,14201
8388608	7,999979	246,9846	127,516	3,141936
9437184	7,999979	270,3573	127,5294	3,143995
10485760	7,999982	255,3956	127,4814	3,141654
15728640	7,999987	291,6301	127,496	3,143619
19922944	7,99999	284,8628	127,5186	3,141409
20971520	7,999992	224,4152	127,4918	3,141268
26214400	7,999992	283,1507	127,5276	3,141438
31457280	7,999994	243,771	127,5057	3,14201
62914560	7,999997	225,1659	127,4941	3,1417
209715200	7,999999	264,4227	127,5002	3,141689
314572800	7,999999	265,5303	127,4986	3,141408
524288000	8	281,2614	127,5017	3,141526
1048576	7,99984	231,9458	127,4971	3,14201
8388608	7,999979	246,9846	127,516	3,141936
9437184	7,999979	270,3573	127,5294	3,143995

As we predicted length has some effect on entropy based algorithm, an average mean, and Monte Carlo Value for Pi, but it is not effective for chi square test and. Furthermore even for 148576 KB input is accurate enough to detect encrypted data and reject other files.

4 Indirect Methods

Main disadvantage of statistical methods when searching for encrypted data is amount of time necessary to search all hard disk area. In contemporary forensic investigations we often have to analyze Terabyte hard drives and it can make days to finish. Forensic investigators cannot afford to use all detection techniques because it would take to long amount of time, that is why we should consider use of some indirect methods which can say as there is high possibility to find encrypted data on the computer. These methods are not supposed to detect encrypted file themselves, but they detect e.g. artifacts of cryptography tools used by system. First technique is to search for any files or records which are related to encrypting tools. Most effective method to search files is to use hash analysis which is one of the basic tool used in computer forensic investigations. In first stage of analysis hash value is computed for every file on evidence disk image. Most often 128 bit Message Digest 5 Algorithm (md5) is used in this purpose. Second stage of hash analysis is to compare computed hash values with hash sets. This helps to exclude files which are not related to case, or highlight those which are related to e.g. malware, illegal software or in our case files created by cryptographic tools.

Next method is to find any traces of cryptographic tools in operation system registry. Even if application is uninstalled there should be remaining keys, forensic investigators can use registry backups (snapshots) to examine registry in specific time line.

Third method is time-consuming but it should be used in the most demanding cases. It is based on performing binary search of unallocated disk area using keywords related to cryptographic tools. If any of cryptographic tools traces are found we can expect that encrypted files can be stored on evidence disk image, and we have good reason to use superior methods of encryption detection based e.g. on entropy.

5 Summary

In the chapter we have explained that in computer forensic investigations one of the most challenging problems is to find encrypted data stored on disk. Most accurate solution would be using all detection techniques presented in this chapter. Unfortunately in most cases we have limited resources and we cannot do it. We think optimal algorithm should consist of hash analysis to exclude known files and to find any files correlated to cryptographic tools. Subsequently indirect method of detecting encryption tools should be used to specify most probable encrypted files types. Next step is to perform signature analysis. As a result we should get all files pointed as encrypted from its signature, and those which doesn't have any known header. Output files should be checked by statistical analysis to confirm or deny that they in very high probability are encrypted files. One problem which can be significant when using statistical algorithms is distortion caused by header of a file. Even if data of file is encrypted, headers usually are in plain text, which can affect result of algorithm in the way that detection algorithm values will drop below detection threshold. Simple defense is to bypass first KB of file. In case of True Crypt it is not necessary because container does not possess its own header. Our tests confirmed that encrypted files are possible to discover using statistical methods based on randomness tests. Most accurate results were given by chi square tests, where values between encrypted and other files where thousand times higher. Surprising was low accuracy of entropy based techniques in differencing encrypted and compressed files. Using entropy would produce a large number of false positive hits. On the other hand compressed files should be excluded from encryption detection by using signature analysis. Our future research will include efficiency issues with large data sets, to choose the most practical statistical algorithm.

References

[1] Hamming, R.W.: Coding and Information Theory. Prentice-Hall, Englewood Cliffs (1980)
[2] Knuth, D.E.: The Art of Computer Programming, Seminumerical Algorithms, vol. 2. Addison-Wesley, Reading (1969)
[3] Ziv, J., Lempel, A.: A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory 23(3), 337–343
[4] Park, S.K., Miller, K.W.: Random Number Generators: Good Ones Are Hard to Find. Communications of the ACM, 1192 (October 1988)

[5] Steve, B.: The Official EnCase Certified Examiner Study Guide. Wiley Publishing, Chichester (2008), ISBN: 978-0-470-18145-4
[6] Liu, V., Brown, F.: Bleeding-Edge Anti-Forensics, InfoSec World (April 3, 2006)
[7] Rogers, D. M.: Anti-Forensic Presentation, Lockheed Martin. San Diego (2005)
[8] Regulation of Investigatory Powers Act 2000, ch.23, UK legislation (July 28, 2000)
[9] Schneier, B.: Rubber-Hose Cryptanalysis. Schneier on Security (October 27, 2008)
[10] Soghoian, C.: Turkish police may have beaten encryption key out of TJ Maxx suspect. Surveillance State, CNET Networks (October 24, 2008)
[11] Huebnera, E., Bema, D., Wee, C.K.: Data hiding in the NTFS file system. Digital Investigation 3(4), 211–226 (2006)
[12] Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
[13] Eoghan, C., Stellatos, G.J.: The impact of full disk encryption on digital forensics. ACM SIGOPS Operating Systems Review 42(3) (April 2008)
[14] Hamming, R.W.: Coding and Information Theory, 2nd edn. Prentice-Hall, Englewood Cliffs (1986)
[15] Haahr, M.: An Introduction to Randomness and Random NumbersRandom (June 1999); Random.org
[16] Walker, J.: Introduction to Probability and Statistics. A Pseudorandom Number Sequence Test Program, Fourmilab (January 28, 2008)
[17] Marco, S.G.: Corresponding The birth of a new industry: entry by start-ups and the drivers of firm growth: The case of encryption software. Research Policy 33(5), 787–806 (2004)

Metric-Probabilistic Assessment of Multi-Version Systems: Some Models and Techniques

Vyacheslav Kharchenko[1,2], Andriy Volkoviy[1,2], Olexandr Siora[1], and Vyacheslav Duzhyi[2]

[1] Research and Production Corporation "Radiy",
Ukraine, 25006, Kirovograd, 29, Geroyev Stalingrada str.
email: research@radiy.com
[2] National Aerospace University "KhAI",
Ukraine, 61070, Kharkiv, 17, Chkalov str.
email: v.duzhy@csac.khai.edu

Abstract. This chapter presents a set of models of multi-version systems and related techniques of diversity level and multi-version systems safety assessment. Spectrum of concepts related to common cause failure (CCF) is expanded, as well as models of multi-version systems (MVS) and multi-version projects (MVP). Approach to metric-probabilistic assessment is proposed and considered in the context of evaluating the software-based and FPGA-based MVS. Direct and indirect estimation of diversity with metrics are considered as parts of system safety or reliability assessment with relevant reliability models. Comparative analysis of the reliability of redundant multi-channel systems of different architectures with the option of diverse components is carried out.

1 Introduction

To ensure reliability, safety and other attributes of dependability of critical computer-based systems and to decrease a probability of common cause failure (CCF) diversity approach is used. This approach implies creation of a few diverse design options of redundant channels (chains which include receiving input data, logic and output signals generation) or parts of the channels (hardware-software products) [1]. Application of product-process diversity is a severe requirement of national and international standards and actual practice in safety engineering [2,3].

The probability of common cause failure and effect of diversity may be essentially decreased due to combined usage of a few of version redundancy kinds on the assumption of maximal independence of redundant channels (designed versions). Application of the modern IT and electronic technologies, on the one hand, improve dependability characteristics of the critical computer-based systems, but on the other hand, some technologies cause additional risks or safety deficits. For example, the advantages of microprocessor technology are well-known; however, a program realization may increase CCF probability of complex software-based

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 87 – 100.
springerlink.com

systems [1,4]. Software faults for microprocessor-based systems and design faults for any systems, including FPGA-based, are the most probable reason of CCFs, because these faults are replicated in redundant channels. In fact, redundant systems with identical channels are non-tolerant to design faults.

There are two main problems in the area of diversity approach application: choice of product-process diversity kinds and assessment of multi-version systems (MVS) safety. In our opinion, inaccurate evaluation of actual level of diversity (and system safety in whole) is a key challenge. If the assessment result is underestimated, it causes additional efforts, costs and increases time of implementation. If the assessment result is overestimated, it causes inadmissible increasing of CCF risks. Methods of diversity level assessment and evaluation of MVS dependability and safety were analyzed in [3-5].

Set-theoretical and metric-oriented methods are based on: (1) set diagrams (Venn diagrams) for design, physical and interaction faults and vulnerabilities of versions and units of voting and reconfiguration; (2) matrix of diversity metrics for individual, group and absolute faults of versions; (3) calculation of diversity metrics using set diagrams and other data of versions testing. Probabilistic methods use reliability block-diagrams (RBDs), Markovian chains (MC), Bayesian method, etc. The RBD and MC-based methods were developed and researched for typical duplicated and majority multi-version architectures. Statistical methods include: gathering and normalization of version fault trends using testing and operation data; choice of software reliability growth model (SRGM) taking into account product and processes features and fitting SRGM parameters; metrics diversity assessment; calculation of reliability and safety indicators. Fault injection-based assessment consists of: receiving project-oriented fault profiles; performing of faults injection procedure; proceeding of data and metrics diversity calculation; calculation of reliability and safety indicators. Expert-oriented methods use two groups of metrics: direct diversity metrics; indirect diversity metrics.

Objective of the chapter is generalization of multi-version systems models and development of some procedures for metric-probabilistic safety assessment technique for software- and FPGA-based systems.

2 Multi-Version Systems and Common Cause Failures: Elements of Taxonomy

2.1 Multi-Version Computing

The concept of multi-version computing, which is a part of dependable computing based on the use of diversity approach, includes the following elements [5]: *version* – an option identical task (product or process) realization in different ways; *version redundancy* (VR) – a kind of redundancy in which different versions are used; *diversity or multiversity* (MV) – the principle providing the use of several versions and performance of the same function (realization of a product or process) by two and more options; *multi-version system* (MVS) – a system in

which a few versions-products are used; *multi-version technology* (MVT) – a set of the interconnected rules and design actions in which in accordance with MV strategy a few versions-processes leading to development of two or more intermediate or end-products are used; *multi-version project* (MVP) - a project in which the multi-version technology is applied (version redundancy of processes is used) leading to creation of one- or multi-version system; *multi-version life cycle* – a collection of interconnected processes and products realized by use of version redundancy and ensuring development of MVS (or one-version system using MVT) according with specification; *strategy of diversity* – a set of general criteria and rules for selection of MVTs; *diversity metric* – a indicator of local process/product diversity level or summarized MVS diversity level.

There are a few concepts for multi-version computing performed by use of two or more kinds of product-process version redundancy (principle of *multi-diversity* or "*diversity of diversity*" (Di2)) in frameworks of the system. Every kind of diversity is peculiar echelon of defence in depth decreasing risks of CCFs.

2.2 Common Event and Common Cause Failures

CCF is an event when e_f (two or more) channels (versions) of redundant *e*-channel (*e*-version) system fail simultaneously and there is a common reason caused this event. Thus, CCF is a multiple failure (MF). It is an alternative of a single failure (SF). On the other hand, multiple failures occur as a result of not only one (common) cause. Multiple failures may be caused by influence of a few different reasons if these reasons concur or spread of influence time values is less than speed of on-line testing and reconfiguration means. In this case MF may be called a common time failure (CTF). Hence, CCF and CTF are multiple failures or common event failures (CEF).

Attributes of the classification form simple hierarchy. CCFs and CTFs may be additionally divided in two groups in accordance with a number of failures (partial and full CCFs, i.e. PCCFs and FCCFs, and partial and full CTFs, i.e. PCTFs and FCTFs) and distinguishability of channel output data on failures, i.e. distinguishable (DCCFs, DCTFs) and undistinguishable (UDCCFs, UDCTFs) failures.

Authors of works related to NPP safety problems, first of all, attend to CCFs analysis. However, CTFs are the important objective of research as there are examples of serial failures caused by attacks on vulnerabilities of redundant channels and other reasons. Besides, a very important problem, in our opinion, is the analysis of distinguishability of effect failures, because it allows determining the moment of partial or full CCFs (or CTFs) by simple means of channel output data comparison.

2.3 Diversity Metrics: β-Factor

To assess a probability of common cause failure it is necessary to calculate the metrics for different CCF vulnerabilities (Fig. 1). Circles of these diagrams correspond to sets of version defects (faults) causing failure. For one-version (one-channel) system

(Fig. 1,a) a number of faults equals N ($N = Card\ F$) and any fault of set F is fatal (equivalent of CCF). In this case metric of CCF β determining relation of number of CCFs to total number of failures equals 1 (and $\alpha = \beta = 1$).

For two-version system (Fig. 1,b,c) CCF metric $\beta = N_{CCF} / N$, $N_{CCF} = Card F_1 \cap F_2$; value of N may be calculated as an arithmetic mean $N = (N_1 + N_2) / 2$, $N_i = Card\ F_i$; SF metric $\alpha_i = 1 - \beta$; DCCF metric $\beta_d = N_{DCCF} / N$; UDCCF metric $\beta_{\bar{d}} = N_{UDCCF} / N$; $\beta = \beta_d + \beta_{\bar{d}}$. Besides, it is possible to use metrics of relative number of DCCFs and UDCCFs: $\beta_d^* = \beta_d / \beta$, $\beta_{\bar{d}}^* = \beta_{\bar{d}} / \beta$.

For three-version system (Fig. 1,d) $\alpha = 1 - \beta - 2\gamma$, where γ is PCCF metric (metric determining part of CCFs of any two versions, $\gamma = 2N_{PCCF} / N$). Metrics of distinguishable and undistinguishable PSSFs are calculated by analogy β_d and $\beta_{\bar{d}}$. If $\gamma = 0$ (Fig. 1,e), $\alpha = 1 - \beta$. This approach is based on the results described in [6] and may be extended to systems in which a set of faults is added a set of vulnerabilities attacked by external system. In the Section 4 metric-based assessment will be used to calculate probability of CCF and MDVS safety indicators.

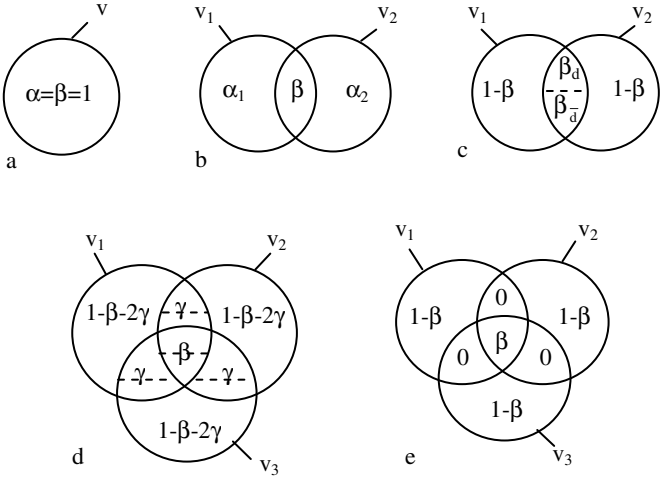


Fig. 1 Diagrams of failures of one-version (a), two-version (b,c) and three-version (d,e) systems

3 Models of MVSs

3.1 W(n): Simplest MVS

One-version *W(1)* and multi-version *W(n)* systems are defined by 4 and 6 variables:

$$W(1) = \{X, Y, Z. \Phi\}, \quad (1)$$

$$W(n) = \{X, Y, Z. \Phi, V, \Psi\}, \quad (2)$$

where X, Y, Z – sets of input signals, internal conditions (states) and output signals correspondingly; $\Phi = \{\varphi_i, i=1, ..., a\}$ – a set of I&C functions (for examples, actuation functions or algorithms of reactor trip system); $V = \{v_j, j=1, ..., n\}$ – a set of versions with output signals $Z_1,\ldots, Z_n$ (or signals Z_{id}, $d = 1,\ldots, n_i$; n_i is a number of versions for function φ_i; $\forall \varphi_i \sim v_j = \{ v_{ij}, j =1,...,n_i\})$; $\Psi = \{\psi_s, s=1, ..., в\}$ – mapping $Z_i \rightarrow Z$.

If the function φ_i is performed, local mapping is true: $\psi_s:\{z_i(v_{i1}),..., z_i(v_{in_i})\}\rightarrow Z_i^{(S)}$. Taking into account formulas (1) and (2), multi-version system and one-version system are connected by relationship:

$$W(n) = \{W(1), V, \Psi\}. \quad (3)$$

System $W(1)$ may be structure-redundant and contain usual means Ψ for signals processing from identical channels (versions). In this case *card* V=1. For system $W(n)$ is true that: $\forall j = \overline{1,a} : \exists j: n_i>1$.

Mapping ψ_s is generally described by: a subset of versions $\Delta v_s \subset v_j$ for receiving output signal Z_i; a vector $\vec{t}_s$ of version v_{ij} initialization time ($\vec{t}_s = \{t(v_{i1}),..., (v_{in_i})\}$); a mean of transforming η_s values $z_i(v_{i1}),..., z_i(v_{in_i})$ in output signal Z_i^S. Hence, $\forall \psi_s \in \Psi: \psi_s = \{ \Delta v_s, \vec{t}_s, \eta_s\}$ and $Z_i^{(S)} = \eta_s [z_i(v_{ij}), \vec{t}_s], v_{ij} \in \Delta v_s$.

There are the following means of transforming η_s: (a) the conjunctive, when $Z_i^S = \mathrm{V} z_i(v_{ij})$; (b) the time conjunctive, when $Z_i^S = \mathrm{V} z_i(v_{ij}) \sigma_{ij}$, where σ_{ij}=1, if $t=t(v_{ij})$, and if not σ_{ij}= 0; (c) the majority, when $Z_i^S = M[z_i(v_{ij})]$, where M – a majority function k out of l (or k out of n); (d) the majority-weighted, when weights of versions $\omega(v_{ij})$ are additionally defined on majorization; (e) the functional, when $Z_i^S = f[z_i(v_{ij})]$, where f – some function of transforming output signals of every version.

3.2 W(n,m) and W(n,m,l): Multi-Diversion Systems

The model (2) describes system with n versions that $n = \sum_{i=1}^{a} n_i$. This model does not take into account the possibility of applying several diversity kinds. A set of version redundancy kinds $R=\{r_d, d=1,..., m\}$ may be decomposed on subsets for versions of products $v_{prd}(t_j)$ and processes $v_{prc}(t_j)$: $R=(\bigcup_j \Delta R_{prdj}) \cup (\bigcup_j \Delta R_{prcj})$, where ΔR_{prdj} and ΔR_{prcj} – appropriate subsets.

Thus, different diversity kinds, $r \in R$, are accumulated in final versions of a multi-version system. It is described by special mapping $\Theta : R \to V$. Mapping Θ may be presented by Boolean matrix $\| \Theta_{dj} \|$, $d = \overline{1,m}$, $j = \overline{1,n}$, where $\Theta_{dj} = 1$, if diversity kind r_p is used in version v_j, and if not $\Theta_{dj} = 0$. Then multi-version system $W(n,m)$ or multi-diversion system is described by formula:

$$W(n,m) = \{ X, Y, Z, \Phi, V, \Psi, R, \Theta \} = \{ W(n), R, \Theta \} = \{ W(1), V, \Psi, R, \Theta \}. \quad (4)$$

It is important to describe correspondence between a set of versions V and a set of redundant channels $C=\{c_q, q=1,...,l\}$. This correspondence may be defined by mapping Q: $V \to C$. This mapping is presented by Boolean matrix $Q = \| \omega_{jg} \|$, $d = \overline{1,m}$, $g = \overline{1,l}$, where $\omega_{gj} = 1$, if version v_i is realized by channel c_j, and if not $\omega_{gj} = 0$. Then model of multi-version (multi-diversion) system is the following:

$$W(n,m,l) = \{ X, Y, Z, \Phi, V, \Psi, R, \Theta, C, Q \} = \{ W(n,m), C, Q \}. \quad (5)$$

3.3 General Model of MVS

MVSs with temporal redundancy and p iterations of algorithms are indicated as $W(n,m,n,p)$ dividing number of parallel (structural) versions n_c, and sequential versions realized by using one channel. Set X may be decomposed for different versions if

$$X = \bigcup_j X_j, \ \forall j_1 j_2 \in \overline{1,n}, \ j_1 \neq j_2 : \ X_{j1} \cap X_{j2}, \ X_{j1} \cap X_{j2} = \varnothing.$$

Such MVSs are called multi-version systems with naturally divided input alphabet:

$$W_{NX} = \{ \{X_j\}, Y, Z, \Phi, V, \Psi, R, \Theta, C, Q \}. \quad (6)$$

If versions process data presented in different notations, such MVSs are called multi-version systems with artificially divided input alphabet W_{AX}. A special function-transformer Π_x (Π_{xj}) should be specified in addition to alphabet X:

$$W_{NX} = \{X, \{\Pi_{xj}\}, Y, Z, \Phi, V, \Psi, R, \Theta, C, Q\}. \quad (7)$$

4 Metric-Probabilistic Assessment of MVS Safety

4.1 General Approach to Metric-Probabilistic Assessment

The proposed approach to assessment of diversity level and MVS safety is based on the following basic procedures analysis and evaluation:

- check-list-based analysis of applicable diversity types (CLD); initial data for the CLD analysis are I&C design and documentation, a table of diversity types (subtypes) was developed in advance; a result of the CLD analysis is a formalized structured information about used diversity types and subtypes in analyzed I&C system;

- metric-based assessment of diversity (MAD); initial data for the MAD procedure are results of the CLD analysis and values of metrics and weight coefficients for diversity types (subtypes) used in I&C systems; a result of the MAD assessment is a value of general diversity metric;
- RBD and Markovian model-based assessment (RDM); initial data for the RDM procedure are I&C design and documentation, results of the CLD and MAD analysis; results of the RDM procedure are values of safety and dependability indicators.

4.2 Assessment of FPGA-Based MVS

The main stages and operations of diversity analysis and MVS assessment depend on the type of the evaluated system. The following description takes into account the peculiarities of FPGA-based systems.

The first stage is a Check-list-based analysis of MVS design and documentation. This stage contains two operations:

1) Analysis of I&C specification and requirements to system, definition of system safety class; requirements to diversity (necessary for diversity application);

2) Analysis of I&C design and development process that involves activities: (a) identification of MVS types: which of the subsystems are FPGA-based and which are software and microprocessor-based; (b) identification of product diversity; for FPGA-based MVSs: manufacturer of chips; FPGA technology; FPGA families; FPGA chips, languages; tools, etc); (c) identification of process diversity kinds.

Results of analysis are entered in a check-list in accordance with rule Yes (if corresponding diversity type is used in a system) / No (in opposite case) and is presented as a n-bit Boolean vector.

The second stage is a metric-based assessment of diversity:

1) Determination of metric values for different types of applied diversity, i.e. performing two activities: (a) determination of metric values (local diversity metrics μ_i for diversity type d_i and local diversity metrics μ_{ij} for diversity subtype d_{ij}); the metric values may be predefined; (b) correction of metric values in accordance with development and operation experience.

2) Calculation of general diversity metric μ for a system: (a) determination (correction) of weight coefficients ω_i (ω_{ij}) of metrics (taking into account multi-diversity aspect); sum of weight coefficients ω_i (ω_{ij}) is equal 1; (b) convolution (additive or more complex) of metrics and calculating value of general diversity metric $\mu = \Sigma\, \omega_i \Sigma\, \omega_{ij}\, \mu_{ij}$, $i = 1,\ldots, n$; $j = 1,\ldots n_i$.

Thus, result of this stage is a value of general diversity metric μ, which is some approximation of β, and can characterize the diversity effect on CCF probability.

4.3 Assessment of Software-Based MVS

The metric-based assessment of software-based MVS can be made using *direct* metrics. General assessment technique of software-based MVS is considered by the example of two-version projects.

To assess diversity indicator β, using *direct* metrics, testing results of each program-version in MVS are required. Direct metrics-based assessment of diversity indicator β of two-version design has the following stages: (1) testing each program-version on the common test set; (2) error determination common for both program-versions; (3) diversity indicator determination by formula:

$$\beta = \frac{2 \cdot n_{com}}{n_1 + n_2},$$

where n_{com} – a number of errors common for both program-versions;

n_1, n_2 – a number of errors in the first and the second program-version, respectively.

In accordance with the formula, diversity indicator β changes from 0 to 1 and takes on limit values in the following cases:

$$\beta = \begin{cases} 0, \text{if } n_{com} = 0 \\ 1, \text{if } n_{err1} = n_{err2} \end{cases}.$$

Gain, obtained by diversity, is $\Delta = 1 - \beta$.

If all errors match in both program-versions (β=1), there will be no gain (Δ=0), because both MVS versions will operate inaccurately. If errors differ in each version (ideal case) (β=0), majority element will be able to determine different values in each channel; so, in this case there will be the largest gain (Δ=1). If diversity indicator is in the range ($0<\beta<1$), obtained gain indicates that in both versions number of undetected errors is decreased by value (Δ*100) percentagewise.

Indirect metrics-based assessment of diversity indicator β of two-version design has the following stages: (1) measurement of absolute values of each program-version metrics, using statistical code analyzer; (2) calculation of absolute value of remainder obtained from a pair of each program-version metrics; (3) rating absolute values of metrics obtained at stage 2; (4) determination of diversity indicator β.

Further, values of diversity indicators, obtained by using direct and indirect metrics, should be compared to determine their correlation.

To assess proposed MVS assessment techniques two-version projects were obtained in programming languages C#, Java, C++. Only versions of initial programs with errors were assessed. Assessment results of two-version projects are presented in Table 1. Each project is a solution for one of the five tasks characterized by complexity level, where I – the lowest level of task complexity, III – the highest level [7]. From the table it appears:

- diversity allows increasing quality of most projects;
- subject diversity allows increasing project quality independently of programming language;
- using diversity for solving complex tasks (Levels of complexity II and III), gain turns to be larger than for simple ones (more than 90%).

Table 1 Results of MVP experimental researches

Task	Level of complexity	Language	Number of projects (MVP)	β			Gain by diversity, % by MVP
				0	0...1	1	
1	I	C#	3	2	0	1	67
		Java	91	16	61	14	85
		C++	465	99	343	23	95
2	II	C#	21	0	21	0	100
		Java	153	12	132	9	94
		C++	465	20	439	6	99
3	I	C#	45	6	38	1	98
		Java	45	0	38	7	84
		C++	435	64	323	48	89
4	II	Java	3	2	1	0	100
		C++	231	0	211	20	91
5	III	Java	3	0	3	0	100
		C++	10	4	6	0	100

5 Probabilistic Assessment of MVS Safety

5.1 Reliability Models of MVS

Probabilistic assessment is considered in terms of Two-channel Reactor Trip System with three parallel tracks (sub-channels) of voting logic "2-out-of-3" in each independent channel. A real system produced by RPC Radiy was taken as a basis [8]. Each of the channels of the system independently receives inputs and form outputs.

A simplified diagram of components of this system is shown in Fig. 2, where $T_{i,j}$ is a track j in channel i. A reliability block diagram of Two-channel System that does not use diversity (channel diversity) is shown in Fig. 3,a. This diagram does not take into account element of voting logic "1-out-of-2" (element OR in the simplest case).

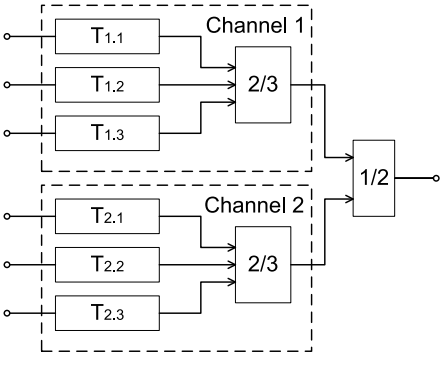


Fig. 2 Simplified structure of Two-channel Three-track System

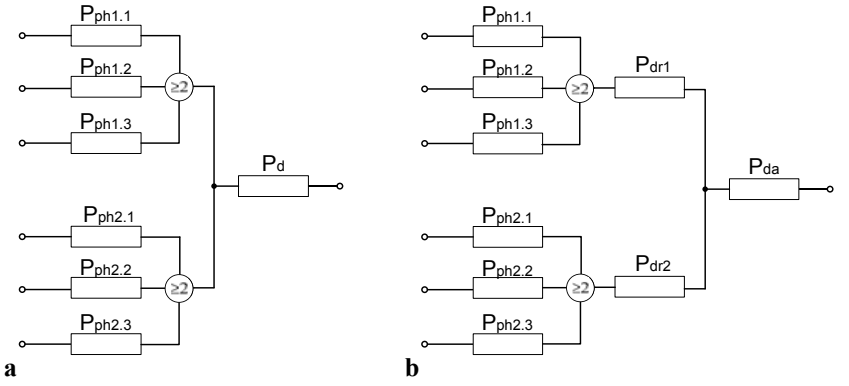


Fig. 3 Reliability Block Diagrams of Two-channel Redundant System

A reliability index $P_{phi.j}$ determines HW reliability of the track *Ti.j* (defined, first of all, by physical failures). Reliability index P_d determines reliability defined by design faults, which may be the main source of common cause failures (CCF). Majority elements have reliability index P_M. Reliability of the One-version Majority Redundant System is represented by the following formula:

$$P_{1D} = \left\{1 - \left[1 - \left(3P_{ph}^2 - 2P_{ph}^3\right)P_M\right]^2\right\}P_d \,. \tag{8}$$

If channels are implemented in different HW and SW versions, value of P_d will consist of three components (see Fig. 3,b):

- $P_{dr1} = 1 - Q_{dr1}$, where Q_{dr1} – a probability of failure caused by relative design faults of the first version;
- $P_{dr2} = 1 - Q_{dr2}$, where Q_{dr2} – a probability of failure caused by relative design faults of the second version;
- $P_{da} = 1 - Q_{da}$, where Q_{da} – a probability of failure caused by absolute design faults (common faults of the versions).

Reliability of Diverse System is calculated by the formula:

$$P_{2D} = \left\{1 - \left[1 - \left(3P_{ph}^2 - 2P_{ph}^3\right)P_{dr}P_M\right]^2\right\}P_{da} \,. \tag{9}$$

We consider that $P_{dr1} = P_{dr2} = P_{dr}$ and majority elements are equally reliable.

Diversity is usually applied in such a configuration, where different channels are independently implemented with different types of diversity. However, this is not the only variant of the redundant circuit. A variant of using redundancy in tracks of one channel is shown in Fig. 4.

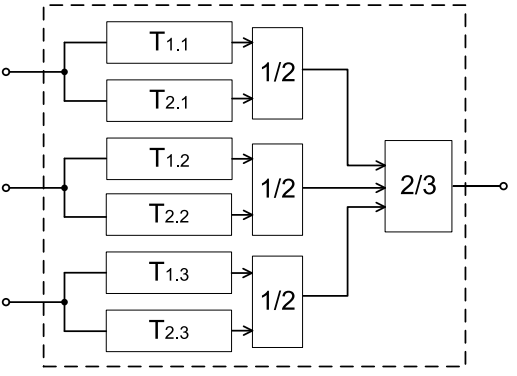


Fig. 4 Simplified structure of Single-channel Three-track System

Reliability block diagrams for the system, represented in Fig. 4, are shown in Fig. 5. Reliability of such system, that use one version for redundancy (Fig. 5,a), can be described by the formula:

$$P_{1M} = \left\{3\left[1-\left(1-P_{ph}\right)^2\right]^2 - 2\left[1-\left(1-P_{ph}\right)^2\right]^3\right\} P_M P_d \,. \tag{10}$$

In case of using two different versions for $T_{1.i}$ and $T_{2.i}$, system has RBD, shown in Fig. 5,b, and a formula for reliability calculation:

$$P_{2M} = \left\{3\left[1-\left(1-P_{ph}\right)^2\right]^2 - 2\left[1-\left(1-P_{ph}\right)^2\right]^3\right\} P_M \left[1-\left(1-P_{dr}\right)^2\right] P_{da} \,. \tag{11}$$

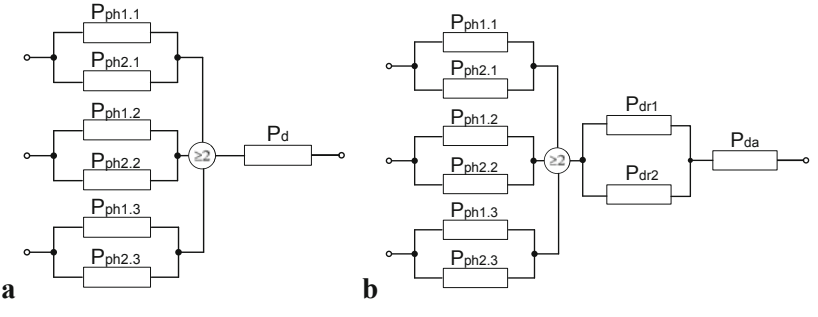


Fig. 5 Reliability Block Diagrams of Single-channel Three-track Redundant System

5.2 MVS Reliability Analysis

If we express the values of reliability (probability of no-failure operation) through failure rates as $P = e^{-\lambda t}$, we can calculate and compare the values of reliability for certain values of λ_{ph}, λ_d, λ_M, λ_{dr}, λ_{da} and β (the fraction of absolute design faults).

Dependence of P_{1D}, P_{2D}, P_{1M} and P_{2M} (formulas (8),(9),(10) and (11)) on the time is graphically shown in Fig. 6. In the calculations the following values of the failure rate were used $\lambda_{ph}=10^{-4}$ 1/h, $\lambda_d=\lambda_{ph}/2$, $\lambda_M=\lambda_{ph}/100$, $\lambda_{di}=(1-\beta)\times\lambda_d$, $\lambda_{da}=\beta\times\lambda_d$, where β=0,1.

Fig. 7 shows how fraction of absolute design faults (FADF is β) effects on reliability of single-channel divers system.

Fig. 8 shows dependence of $\Delta P_{2M\text{-}1M}$ (the difference of probabilities P_{2M} and P_{1M}) and $\Delta P_{2M\text{-}2D}$ (the difference of probabilities P_{2M} and P_{2D}) on time.

It should be noted that, although the single-channel two-version three-track redundant system has the greater effect of the use of diversity, its application in many ways violates the principle of independence. Therefore, the use of such architecture for safety systems of nuclear power plants is complicated.

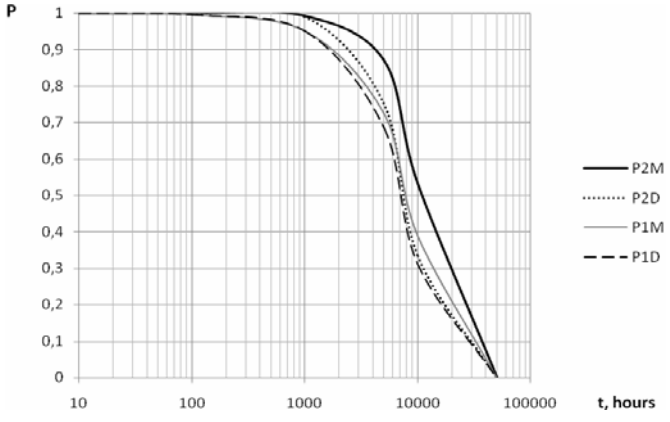


Fig. 6 Dependence of P_{1D}, P_{2D}, P_{1M} and P_{2M} on the time (for systems with diversity β=0,1)

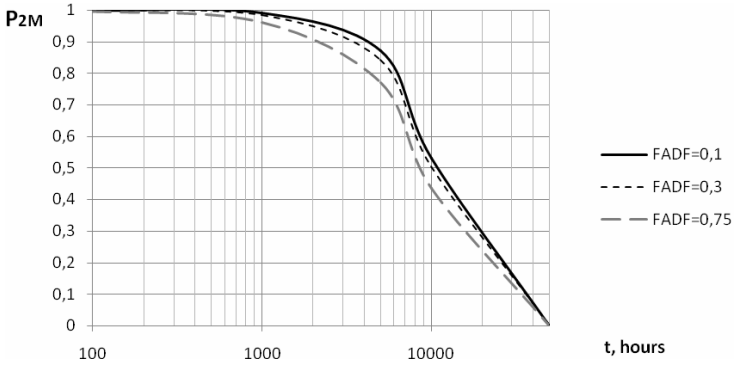


Fig. 7 Dependence of P_{2M} on time and β

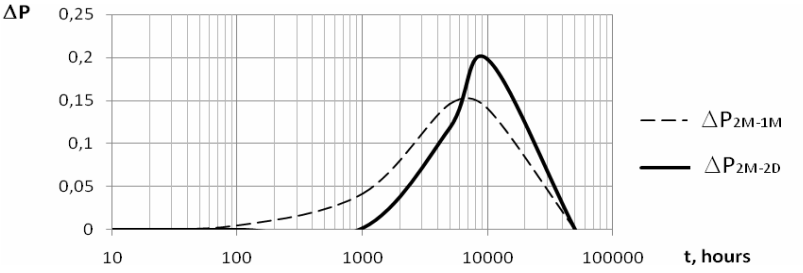


Fig. 8 Dependence of $\Delta P_{2M\text{-}1M}$ and $\Delta P_{2M\text{-}2D}$ on time

6 Conclusion

In this work we discussed some problems regarding to diversity approach application, decreasing probability of common cause failures and assessment of multi-version systems safety. One of the main challenges in this area is a fact that multi-version systems are still unique, failures occurred very rarely, information about failures is not enough representative and is not generalized taking into account development and operation experience for different applications.

Described models of multi-version systems (formulas (2), (4)-(7)) are a base for the development of different architecture variants. The proposed techniques of diversity level and multi-version systems safety assessment are founded on two interconnected approaches. First of them is the metric-based technique allowing to assess a diversity level and to compare multi-version systems on application of different kinds and different volume of diversity. Second one is based on the probabilistic models, which include β calculated using metric analysis.

These theoretical issues were used on development and assessment of Radiy™ Platform-based instrumentation and control systems safety related to NPPs. Next steps of research and development activities may be connected with creation and implementation of tool-based support of all life cycle processes for multi-version systems including decision making related to choice of diversity kinds taking into account results of qualitative and quantitative assessment.

References

[1] Pullum L.: Software Fault Tolerance Techniques and Implementation, AHC Library (2001)

[2] Wood R., Belles R., Cetiner M., et al.: Diversity Strategies for NPP I&C Systems, NUREG/CR-7007 ORNL/TM-2009/302 (2009)

[3] Kharchenko, V., Siora, A., Sklyar, V., et al.: Multi-Diversity Versus Common Cause Failures: FPGA-Based Multi-Version NPP I&C Systems. In: Proceedings of the 76th Conference NPIC&HMIT, Las-Vegas, Nevada, USA (2010)

[4] Littlewood B., Strigini L. A Discussion of Practices for Enhancing Diversity in Software Designs. Technical report LS_DI_TR-04, City University, London, Great Britain (2000)
[5] Kharchenko, V., Siora, A., Sklyar, V., et al.: Diversity-Oriented FPGA-Based NPP I&C Systems: Safety Assessment, Development, Implementation, In: Proceeding by 18th International Conference on Nuclear Engineering (ICONE18), Xi'an, China, 10 p (2010)
[6] Hokstad, P., Rausand, M.: Common Cause Failure Modeling: Status and Trends. In: Misra, K.B. (ed.) Handbook of Performability Engineering, pp. 621–640. Springer, Heidelberg (2008), doi:10.1007/978-1-84800-131-2_39
[7] Duzhyi, V., Kharchenko, V., Starov, O., Rusin, D.: Research Sports Programming Services as Multi-version Projects. Radioelectronic and Computer 47, 29–35 (2010)
[8] Kharchenko, V., Sklyar, V. (eds.): FPGA-based NPP Instrumentation and Control Systems: Development and Safety Assessment, RPC Radiy, National Aerospace University KhAI, State Scientific and Technical Center for Nuclear and Radiation Safety, 188 p (2008)

Two-Level Software Rejuvenation Model with Increasing Failure Rate Degradation

Vasilis P. Koutras

Department of Financial and Management Engineering, University of the Aegean,
Kountouriotou 41 st., Chios GR82100, Greece
email: v.koutras@fme.aegean.gr

Abstract. Nowadays computer systems fail mainly due to software faults. Consequently the need of improving software availability and reliability arises. One of the main reasons of software failures is software aging. To counteract aging, software rejuvenation has been recently proposed. The main aim when dealing with rejuvenation is to distinguish the optimal time or conditions to trigger it. Rejuvenation can be performed in two levels, partial and full. In this paper, a software system experiencing resource degradation is considered and according to the level of the degradation, partial or full rejuvenation is triggered. Since software performance degrades in time due to the increasing resource exhaustion it is proposed to model the degradation time by an increasing failure rate distribution. The system is modeled by a Semi-Markov process. The purpose is to examine how availability and downtime cost are affected by this fact and moreover to decide on the optimal rejuvenation policy.

1 Introduction

The fact that the performance of computer systems can be affected by malfunctions or outages consisting in either hardware or software failures is common knowledge. It is equally well known that hardware failures can be severe and can have undesired consequences not only for computer systems but also for humans. Today however, hardware is considered reliable to a great extent; moreover various mechanisms have emerged which ensure high levels of hardware performance.

On the other hand, software failures are equally or more severe than hardware failures for a computer system. Nowadays, thanks to the evolution of hardware fault tolerance, it is highly unlikely for computer systems to fail because of hardware malfunctions. In opposition to hardware-originated problems though, fault avoidance by appropriate implementation of software engineering practices is still an issue, especially with complex software systems. Moreover, it is almost impossible to fully test and verify that software is fault free. Consequently, software faults are the principal cause of computer systems' failures [2], [6], [21].

One of the main reasons of software failures is software aging. As software systems run continuously, error conditions generated by aging related bugs can be

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 101 – 115.
springerlink.com

accumulated and lead either to performance degradation which in its turn leads to the software system's crash, or hang failure. This phenomenon, reported in the literature as software aging [10], [15], occurs mainly due to memory leaks, unreleased file-locks, data corruption, storage space fragmentation and accumulation of round-off errors [1], [10], [11].

For counteracting software aging, the concept of preventing error accumulation should be introduced. In view of the non-deterministic nature of aging related failures, it is imperative for the aforementioned concept to include an implementation policy which takes into account that a failure occurrence may be completely stochastic. In [10], Huang et al. were the first to introduce the concept of a proactive technique counteracting software aging. The technique was called software rejuvenation and can be considered a software preventive and proactive maintenance technique. In [10]. rejuvenation is defined as "the periodic preemptive rollback of continuously running applications to prevent software failures in the future". In practice, software rejuvenation is the concept of periodically stopping the running software, cleaning its internal state and restarting it [20]. Cleaning the internal state of software might involve garbage collection, defragmentation, flushing operating system kernel tables and reinitializing internal data structures [10], [23]. The main advantage of software rejuvenation lies in its proactive nature.

Nevertheless, rejuvenation obviously involves an overhead. This can be considered as a major disadvantage but, to its defence, one can claim that rejuvenation involves scheduled downtime as opposed to higher downtime caused by unplanned outages. Consequently, when software rejuvenation is allowed for in view of enhancing system availability, reliability and/or reduce downtime cost, the optimal time (optimal schedule) to perform it has to be determined in advance.

Rejuvenation and the optimal rejuvenation schedule have been extensively studied in the literature. Among the various approaches for software rejuvenation, it is interesting to focus on a certain category consisting in considering two-level rejuvenation. The distinction between these levels lies in the actions taken at each one, in order to release system resources. The first level corresponds to partial and the second to full rejuvenation [8], [12], [26]. Usually, the choice of the level of rejuvenation action depends on the level of software degradation. The difference in rejuvenation levels is also reflected on their effects.

In this paper, a software system experiencing resource degradation is considered and according to the level of the degradation, partial or full rejuvenation is triggered. Most of the studies in the area, model resource exhaustion by assuming that the degradation rate is constant as for example in [10], [18], [19], [24]. Contrarily, Xie et al in [26] consider a more general case in which the time to software degradation follows a general distribution, though in a more realistic case, Tai et al in [22], model the degradation time by a hypo-exponential distribution to capture the fact that due to error accumulation and consequently due to aging, the degradation time should be modelled by an increasing failure rate distribution. In our approach, since software performance degrades in time due to the increasing

resource exhaustion, it is proposed to model the degradation time by a Weibull IFR distribution [1].

Additionally, the proposed model considers the probability that rejuvenation cannot be accomplished properly. This is the scenario of a failed rejuvenation action which indicates an abnormal function during the rejuvenation process. The latter situation may be extremely rare but it may occur under certain conditions and circumstances as for instance when garbage collection problems due to too large applications or applications with severe real time constraints occur [3]. When a failure on the rejuvenation procedure occurs, the software system enters a failure state and needs to be rebooted in order to return to a robust state. The concept of a failed rejuvenation can be met either when partial or full rejuvenation actions need to be performed.

The main aim of this paper is to examine how software availability and downtime cost is affected by the IFR distribution of the degradation procedure and moreover to decide on the optimal rejuvenation policy in terms of availability and downtime cost. Since not all of the transition time distributions are exponential, a semi-Markov process is introduced to model the behaviour of the system under consideration.

2 Software Rejuvenation Model

A software system which is initially in a robust state is considered. Due to error condition accumulation and aging phenomena, its performance degrades in time and enters a medium-efficient software execution phase [26]. Although the system is not as efficient as in the robust state, it is still available. Since the system is operational and it runs continuously in time, the level of performance degradation increases leading the system to a low-efficient execution phases [26]. The low-efficient state is also an available state. Actually, by the aforementioned distinction of states, the degradation procedure has been divided into two phases. Hence, from the initial robust state 0, cf. Figure 1, the system enters the medium- efficient state 1 and since the software performance continues to degrade, the system eventually enters the low-efficient state 2.

2.1 Modeling Software Performance Degradation

Software systems running continuously for a long time reveal a degraded performance and an increased failure occurrence rate. This is the so called phenomenon of software aging. In order to prevent failures due to aging, proactive actions should take place. In the case that software degradation rate is constant, proactive action does not affect the probability of a failure. Aging-related bugs, causing software aging, can cause errors to accumulate over time. These error conditions do not lead immediately to failures; instead aging phenomena occur and the software reveals a degraded performance. Additionally, the total system's runtime can

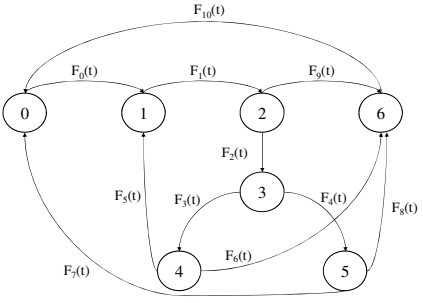


Fig. 1 State transition diagram for the rejuvenation model

affect an aging-related bug's activation rate [9]. Consequently, the degradation rate should be model as an increasing failure rate. In this case, usually in engineering applications, the Weibull distribution is used to model such incidents.

In the present work, it is proposed to model the times to enter a more degraded state by Weibull distributions with shape parameters greater than 1. In detail, the time to enter the medium-efficient state 2, from the robust state, is modelled by a Weibull distribution with shape parameter k_1 ($k_1 > 1$) and scale parameter λ_1. Likewise, the time to enter the low-efficient state 3 from the medium-efficient state is model by the same distribution with parameters k_2 ($k_2 > 1$) and λ_2.

Moreover, in the case that the system is in the medium-efficient state, as it will be explained later on, software rejuvenation should be performed in order to avoid a future software failure. But in the case that software degradation is high, the system may fail and hence enter the failure state 6. Since such a transition is caused due to software gradation, it is also modeled by an IFR Weibull distribution with parameters k_3 ($k_3 > 1$) and λ_3.

Consequently, the software system considered experienced software degradation and failures due to aging phenomena, which are modeled by increasing failure rate distributions. These IFR Weibull distributions are denoted by $F_0(t)$, $F_1(t)$ and $F_9(t)$. Distribution $F_0(t)$ models the time for the system to enter the low-efficient state though $F_1(t)$ correspond to the time for entering the low-efficient state. Finally, $F_9(t)$ is the distribution of the time to enter the failure state from the low-efficient state (cf. Figure 1).

2.2 Two-Level Software Rejuvenation Model

When the software system enters the low-efficient state it is failure prone and software rejuvenation should be triggered in order to prevent a future software failure. The time to trigger software rejuvenation is actually a fixed duration and hence it can model by the unit step function: $F_2(t) = u(t\text{-}r)$ with r denoting the time to trigger rejuvenation [7], [26]. Modeling the time to rejuvenation by the unit step function indicates that the rejuvenation interval (i.e. the time to initiate the rejuvenation procedure) should be fixed and predefined. This means that before r elapses

no rejuvenation action is initiated. After time r the system enters is taken out of service and rejuvenation is initiated.

Software rejuvenation can be offered at two levels depending on the level of software performance degradation. In this paper, rejuvenation is proposed to be triggered according to a two-level model. The distinction between these levels, apart from the level of degradation, lies in the actions taken by each in order to release system resources avoiding hence a probable software failure. The first level corresponds to partial and the second to full rejuvenation [8], [12], [26]. Partial and full rejuvenation ([26]) are defined as follows.

Partial rejuvenation is a service-level rejuvenation. When it is performed, all necessary data is saved and the following restart of the service resumes the operation in a more usable state. It can also be considered as a partial restart. The impact of other applications running on the same operating system (OS) is assumed to be minimal. Partial rejuvenation returns the system to a more usable state but not in an "as good as new" state.

During full rejuvenation the OS and all the services of the machine which undergoes it have to be stopped. In fact, the full rejuvenation action restarts the OS to recover all its free memory, therefore it can be also considered as a full restart. It is a thorough rejuvenation, and it affects all applications on the machine undergoing it. As a consequence, it takes more time to complete it and once it is completed, the system enters a robust state.

Both in the case of partial or full rejuvenation and under certain circumstances there is a probability that something can go wrong during the rejuvenation procedure, resulting in a failure on the system's software. For instance, some problems may occur during garbage collection, especially when dealing with too large applications with severe real time constraints. The above concept called failed rejuvenation has been recently introduced in [12], [13] and [14]. The phenomenon of failed rejuvenation can be considered as a rather rare event but we are interested in the fact that there is a probability, although it is very low, that rejuvenation may fail to be completed and cause a software failure.

2.3 Model Description

Given the degradation, failure and rejuvenation procedure description the behaviour of the software system presented can be described in detailed. Hence a sort description of the system is provided which is in accordance with Figure 1.

Initially the system is at the robust state 0 which is a fully operational state. Due to aging phenomena, the system experiences performance degradation and enters the medium-efficient state 1with an IFR Weibull distribution $F_0(t)$. Since error conditions accumulate over time the system can a experience a more severe degradation and hence enter the low-efficient state 2, again according to an IFR Weibull distribution $F_1(t)$. In this state the system is failure prone. Hence, either rejuvenation is decided and the system enters state 3 with distribution $F_2(t)$ or the system fails with an IFR Weibull distribution $F_9(t)$, entering hence state 6.

In the case of rejuvenation, depending on the level of software performance degradation, the system from state 3 can enter the partial rejuvenation state 4

according to an exponential distribution $F_3(t)$ with parameter r_p or it can enter the full rejuvenation state 5, also according to an exponential distribution $F_4(t)$ with parameter r_f. The system recovers from partial rejuvenation with rate β_p, which is the parameter of the exponential distribution $F_5(t)$, and enters the low-efficient state 1. Recovery from the full rejuvenation state 5 occurs with rate β_f which denotes the parameters of the exponential distribution $F_7(t)$. In this case, the system returns to the highly robust state 0. Both partial and full rejuvenation can fail to be accomplished. When partial rejuvenation fails, the system enters the failure state 6 with rate λ_p while when full rejuvenation fails the corresponding transition rate is λ_f. Notice that the time to a failed rejuvenation action is exponentially distributed for both partial and full rejuvenation ($F_6(t)$ and $F_8(t)$ correspondingly).

In the case that performance degradation is high enough and a software failure occurs before rejuvenation interval r elapses, the system enters the failure state 5 with an IFR Weibull distribution $F_9(t)$.

From the failure state 6, the system needs to be restarted in order to recover to the robust state. The time needed to recover from a failure, is assumed to be exponentially distributed ($F_{10}(t)$) with parameter μ.

In order to avoid any misunderstandings, the distributions of the transition times as they appear in Figure 1, are provided:

$$\begin{array}{lll} F_0(t)=1-e^{-\lambda_1 t^{k_1}} & F_3(t)=1-e^{-r_p t} & F_7(t)=1-e^{-\beta_f t} \\ F_1(t)=1-e^{-\lambda_2 t^{k_2}} & F_4(t)=1-e^{-r_f t} & F_8(t)=1-e^{-\lambda_f t} \\ F_2(t)=\begin{cases}1 & ,t\geq r\\ 0 & ,t<r\end{cases} & F_5(t)=1-e^{-\beta_p t} & F_9(t)=1-e^{-\lambda_3 t^{k_3}} \\ & F_6(t)=1-e^{-\lambda_p t} & F_{10}(t)=1-e^{-\mu t} \end{array} \quad (1)$$

3 Semi-markov Analysis

Since not all of the transitions among the states of the studied system obey in exponential distributions, the evolution of the system has to be studied via a semi-Markov process. The Semi-Markov analysis is given by the so-called two-stage method [5], [25]. Based on this method, the kernel matrix $\mathbf{Q}(t)$ is needed:

$$\mathbf{Q}(t)=\begin{bmatrix} 0 & q_{01} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{23} & 0 & 0 & q_{26} \\ 0 & 0 & 0 & 0 & q_{34} & q_{35} & 0 \\ 0 & q_{41} & 0 & 0 & 0 & 0 & q_{46} \\ q_{50} & 0 & 0 & 0 & 0 & 0 & q_{56} \\ q_{60} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2)$$

where $q_{ij}(t) = Pr\{Y_1 = j, T_1 \le t \mid Y_0 = i\}$, $i, j \in \mathrm{E}$ and E={0, 1, 2, 3, 4, 5, 6} the state space of the model, $\{(Y_n, T_n), n \ge 0\}$ the underlying Markov renewal sequence of random variables [26]. Thus q_{ij} is the probability that the SMP has entered state i, the next transition occurs within time t and the process transits to state j.

Denoting by $F_i(t)$ the cumulative distribution function (cdf) of any transition from state $i \in \mathrm{E}$ and $\overline{F_i}(t) = 1 - F_i(t)$ its complementary cdf, the elements of the kernel matrix **Q**(t) are given as follows:

$$q_{01} = F_0(t), \quad q_{12} = F_1(t), \quad q_{23} = \int_0^t \overline{F_9}(x)\,dF_2(x), \quad q_{26} = \int_0^t \overline{F_2}(x)\,dF_9(x),$$

$$q_{34} = \int_0^t \overline{F_4}(x)\,dF_3(x), \quad q_{35} = \int_0^t \overline{F}_3(x)\,dF_4(x), \quad q_{41} = \int_0^t \overline{F}_6(x)\,dF_5(x), \tag{3}$$

$$q_{46} = \int_0^t \overline{F}_5(x)\,dF_6(x), \quad q_{50} = \int_0^t \overline{F}_8(x)\,dF_7(x), \quad q_{56} = \int_0^t \overline{F}_7(x)\,dF_8(x), \quad q_{60} = F_{10}(t)$$

Recall that the time to trigger rejuvenation is a fixed duration and hence its cdf can be given as $F_2(t) = u(t - r)$, where u(t) is the unit step function and r is the time to trigger rejuvenation [7], [26].

Since such software systems are designed to run continuously we are interested in their asymptotic behavior. In order to derive the asymptotic probability distribution of the embedded Markov chain (EMC) the following linear system of equations

$$\mathbf{v} = \mathbf{vP}, \sum_{i \in \mathrm{E}} v_i = 1$$

has to be solved, where $v = (v_0, \ldots v_6)$ is the steady-state probability vector of the EMC and $\mathbf{P} = \lim_{t \to \infty} \mathbf{Q}(t)$ is the corresponding probability matrix which can be computed based on the two-stage method of SMP analysis:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{-\lambda_3 r^{k_3}} & 0 & 0 & 1 - e^{-\lambda_3 r^{k_3}} \\ 0 & 0 & 0 & 0 & \dfrac{r_p}{r_p + r_f} & \dfrac{r_f}{r_p + r_f} & 0 \\ 0 & \dfrac{\beta_p}{\beta_p + \lambda_p} & 0 & 0 & 0 & 0 & \dfrac{\lambda_p}{\beta_p + \lambda_p} \\ \dfrac{\beta_f}{\beta_f + \lambda_f} & 0 & 0 & 0 & 0 & 0 & \dfrac{\lambda_f}{\beta_f + \lambda_f} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

The steady-state probability of state *i* for the SMP can be computed taking into account the mean sojourn time h_i in state *i*. The asymptotic probability distribution of the SMP exists, since the embedded EMC Y_n is irreducible recurrent [5]. As already mentioned, the distribution of time spend at states 0, 1 and 2, is assumed to be a Weibull distribution with shape parameters k_1, k_2, $k_3 > 1$ correspondingly, indicating that they are IFR distributions. Moreover, the mean sojourn times at state 3, where the system is taken out of service for rejuvenation and the level of rejuvenation has to be decided, is negligible to comparing with the rest of the sojourn times and hence is assumed to be zero. Let $H_i(t)$ denote the distribution of the sojourn time in state *i*. Then the mean sojourn time in state *i* is $h_i = \int_0^\infty \bar{H}_i(t)dt$. Hence:

$$h_0 = \int_0^\infty \overline{F_0}(t)dt = \lambda_1 \cdot \Gamma\left(1+\frac{1}{k_1}\right), \quad h_1 = \int_0^\infty \overline{F_1}(t)dt = \lambda_2 \cdot \Gamma\left(1+\frac{1}{k_2}\right),$$

$$h_2 = \int_0^\infty \overline{F_2}(t)\overline{F_9}(t)dt = \frac{\Gamma\left(1/k_3\right)}{k_3\lambda_3^{1/k_3}} \cdot \Gamma\left(\lambda_3 \cdot r^{k_3}, \frac{1}{k_3}\right), \quad h_4 = \int_0^\infty \overline{F_5}(t)\overline{F_6}(t)dt = \frac{1}{\beta_p + \lambda_p} \tag{5}$$

$$h_5 = \int_0^\infty \overline{F_7}(t)\overline{F_8}(t)dt = \frac{1}{\beta_f + \lambda_f}, \quad h_6 = \int_0^\infty \overline{F_{10}}(t)dt = \frac{1}{\mu}, \quad h_3 = 0,$$

where Γ(•) Gamma function, while Γ(•,•) is the incomplete Gamma function.

3.1 Asymptotic Availability

The main aim of this study is to examine firstly how the IFR Weibull distributions of the degradation and failure procedures affect system's asymptotic availability and downtime cost and secondly to decide on the optimal rejuvenation schedule *r*. The asymptotic availability $A=\Sigma_{i\in U}\pi_i$ has to be derived, with π_i denoting the stationary probability of the SMP for state *i*. Set U is a subset of the state space E constrained on the operational states of the system. The complementary subset of U is subset D, which contains the down states of the system. Analytically, U = {0, 1, 2}, D = {3, 4, 5, 6}. To compute the asymptotic availability, the stationary probability distribution of the SMP $\pi_i = {v_i h_i}/{\sum_{k\in E} v_k h_k}$ is needed. The following formula provides the asymptotic availability:

$$A = \pi_0 + \pi_1 + \pi_2 = \frac{v_0 h_0 + v_1 h_1 + v_2 h_2}{v_0 h_0 + v_1 h_1 + v_2 h_2 + v_4 h_4 + v_5 h_5 + v_6 h_6} \tag{6}$$

As it can be observed by the probability transition matrix in (4) and the mean sojourn times in (5), the steady-state probability distribution of the SMP and consequently the asymptotic availability depends on the Weibull parameters k_1, λ_1, k_2,

λ_2, k_3, λ_3 and on the rejuvenation schedule r. Hence, it is interesting to examine how the variation of these parameters affects system's availability.

3.2 Total Expected Downtime Cost

Apart from the asymptotic availability, we are also interested on the cost caused due to downtime. As it is mentioned, rejuvenation incurs an overhead and consequently it generates a downtime cost since when the system is rejuvenated is a non-operational mode. On the other hand, letting the system without rejuvenation can lead to unexpected software failures due to aging phenomena that cause software performance degradation. Such unexpected outages due to their unplanned nature cost much more that the scheduled rejuvenation actions [10]. A downtime cost occurs when the system enters the states belonging to state subset D.

To compute the downtime cost, a performability indicator is introduced. In the following, notation C_f stands for the failure cost per unit of time while C_1 stands for the partial rejuvenation cost per unit of time and C_2 for the full rejuvenation cost per unit time. As previously stated it holds that $C_f > C_1$, $C_f > C_2$ and additionally $C_f > C_1 + C_2$. According to the definitions of partial and full rejuvenation, the second rejuvenation action is more cost effective than the first, implying $C_1 < C_2$.

For defining the performability indicator, notice that: $E = U \cup D$, $U \cap D = \varnothing$, $U \neq \varnothing$ and $D \neq \varnothing$. Now let

$$d_i = \begin{cases} 1, i \in D \\ 0, i \in U \end{cases} \text{ and } C_i = \begin{cases} C_f, i = 6 \\ C_1, i = 4 \\ C_2, i = 5 \\ 0, \quad i \in E / \{4,5,6\} \end{cases}$$

defining the reward function $w(i) = C_i \cdot d_i$. The cost per unit of downtime is denoted as:

$$g(X(t)) = \sum_{i \in E} w(i) \cdot 1_{\{X(t)=i\}} \tag{7}$$

where $X(t)$ denotes the state of the system at time t. Then the expected downtime cost is given by the following equation ([16]):

$$E[g(X(t))] = E\left[\sum_{i \in E} w(i) \cdot 1_{\{X(t)=i\}}\right] = \sum_{i \in E} w(i) \cdot Pr(X(t) = i) \tag{8}$$

System's performance can be measured by the total expected downtime cost in the long run, hence the Total Expected Downtime Cost (*TEDC*) in the steady state in a time interval of L time units, is given by ([10], [17]):

$$TEDC(L) = \left(\lim_{t \to \infty} E[g(X(t))]\right) \times L = \left(\sum_{i \in E} w(i) \cdot \pi_i\right) \times L \tag{9}$$

which is finally:

$$TEDC(L) = \left(C_f \cdot \pi_6 + C_1 \cdot \pi_4 + C_2 \cdot \pi_5\right) \times L \tag{10}$$

Since *TEDC* depends on the stationary probability distribution of the SMP, it also depends on the Weibull distributions parameters and on the rejuvenation schedule. Hence, the effects of the above parameters on the total expected downtime cost can be also examined.

4 Numerical Illustration

To illustrate the theoretical framework, some numerical results are presented. The data to be used do not come out of a real life system but they are in accordance with the relative literature. All the data needed are shown in Table 1. Notice that the Weibull distribution for the degradation and the failure procedures are considered as IFR. The parameters of the failure time Weibull distribution are fixed.

Contrarily, it is assumed that the degradation time distribution parameters are identical, i.e. in the following $k_1 = k_2$ and $\lambda_1 = \lambda_2$.

The aim is to examine how the Weibull parameters and the rejuvenation schedule affect system's asymptotic availability and the total expected downtime cost. These effects are represented through Figures 2-9.

Table 1 Empirical data

Parameter	Values	Parameter	Values
r_p	0.1104 **days**$^{-1}$	λ_p	0.0016 **days**$^{-1}$
r_f	0.0552 **days**$^{-1}$	λ_f	0.0033 **days**$^{-1}$
β_p	480 **days**$^{-1}$	μ	120 **days**$^{-1}$
β_f	240 **days**$^{-1}$	(k_3,λ_3)	(2,0.8 **days**$^{-1}$)

In Figure 2, it is shown that when the shape parameter of the Weibull distribution increases, the availability decreases. Since the shape parameter is set to be higher than 2, the time to degradation follows an IFR distribution. This means that the probability of software degradation increases with time, which is sensible since error conditions are accumulated increasingly with time. As higher the shape parameter is as intensely the availability decreases. On the other hand, the scale parameters depicting the skewness and the kurtosis of the degradation time distribution, when increased causes an increase on the availability. An increase of the scale parameter means that the probability of transitions to a more degraded state decreases and hence the probability of entering a non- operational state decreases. The same effects of the Weibull parameters can be obtained for the total expected downtime cost, as shown in Figure 3. Higher values for the shape parameter cause an increase on the downtime cost since the probability of entering a non-operational state increases with time. Correspondingly, the total expected cost

decreases for higher values of the shape parameter, since in this case the probability of entering a downstate decreases.

Interesting remarks can be obtained by Figures 4 and 5. Notice that the probability of the system remaining at state 2 decreases for values lower than approximately 1.2 days and then increases. That is why availability and downtime cost show a minimum and a maximum correspondingly for rejuvenation interval around this value. Moreover, it is observed that for rejuvenating the system every 3 days from the time that it enters the low efficiency state, one can manage to achieve five nines availability and simultaneously achieve an important reduction of the downtime cost. Consequently, a 3-day rejuvenation interval can be considered as an optimal schedule.

All of the aforementioned effects hold true, when combining the rejuvenation schedule and the Weibull parameters, in order to examine how they affect availability and downtime cost. The corresponding plots are shown in Figures 6-9.

It is worth mentioning that all the observation taken out of the graphs are data sensible. Nevertheless, the remarks about the role of rejuvenation schedule and Weibull parameters on the availability and on the downtime cost do not change. The shape of availability and downtime cost in these figures is highly affected by the rejuvenation interval. Consequently, availability is convex around rejuvenation intervals of about 1.2 days and downtime cost is concave correspondingly, in accordance with Figures 4 and 5, as previously explained.

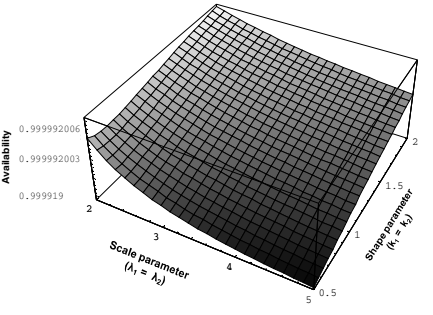


Fig. 2 Availability vs. Weibull parameters

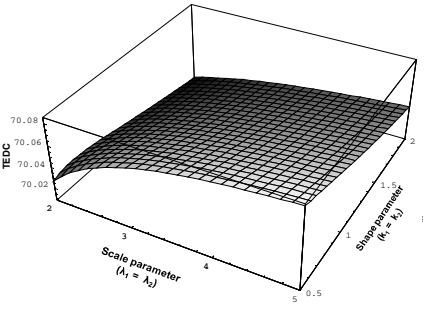


Fig. 3 TEDC vs. Weibull parameters

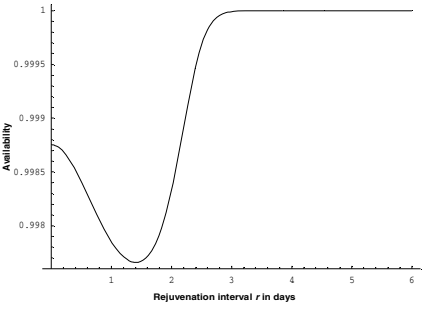


Fig. 4 Availability vs. Rejuvenation interval

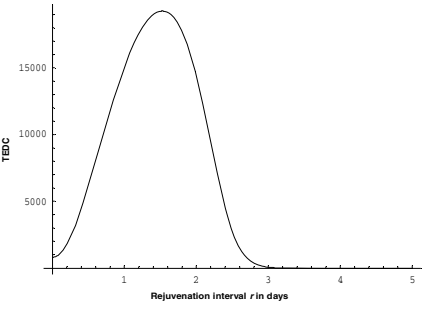


Fig. 5 TEDC vs. Rejuvenation interval

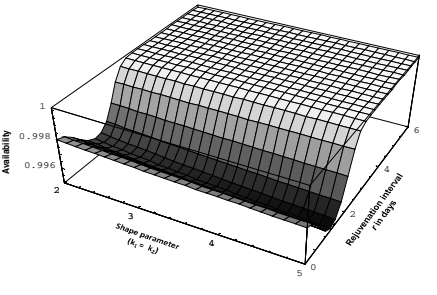


Fig. 6 Availability for varying shape parameter and rejuvenation interval

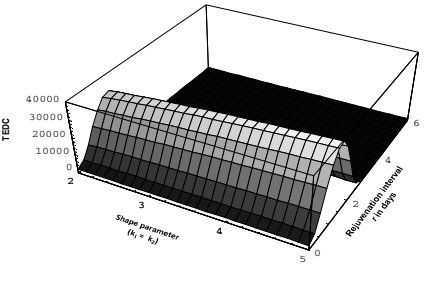


Fig. 7 TEDC for varying shape parameter and rejuvenation interval

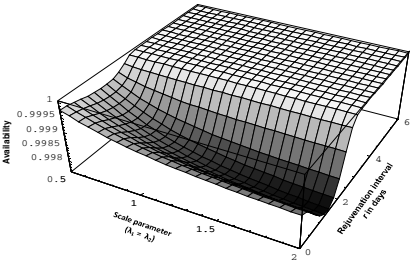


Fig. 8 Availability for varying scale parameter and rejuvenation interval

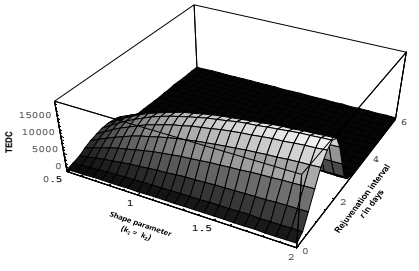


Fig. 9 TEDC for varying scale parameter and rejuvenation interval

5 Conclusions

In this paper a software system experiencing performance degradation in time is considered. To prevent software failures due to degradation software rejuvenation is adopted in two levels. The innovative aspect of the present study consist in modelling the degradation time with an increasing failure rate distribution, in order to highlight the fact that the probability of degradation increases with time. Hence a two-parameter Weibull distribution is considered for the two-levels of software performance degradation. Along with distinguishing the optimal rejuvenation schedule that benefits asymptotic availability and the expected downtime cost, the effect of the degradation time distribution parameters on these measures are also examined. Consequently, the theoretical framework, illustrated by a numerical example, of a two-level software rejuvenation model with increasing failure rate degradation is introduced, extending hence the existing literature on the area. In the future, it might be interesting to adopt some other distributions to model the degradation procedure and examine how they affect the dependability of the two-level rejuvenation model.

References

[1] Bae, S.J., Kuo, W., Kvam, P.H.: Degradation models and implied lifetime distributions. Reliability Engineering and System Safety 92(5), 601–608 (2007)
[2] Bobbio, A., Sereno, M.: Fine grained software rejuvenation models. In: Proceedings of IEEE International Symposium on Computer Performance and Dependability, IPDS 1998, pp. 4–12 (1998)
[3] Boehm, H., Weiser, M.: Garbage collection in an uncooperative environment. Software-Practice and Experience 18, 807–820 (1988)
[4] Casteli, V., Harper, R.E., Heidelberger, P., Hunter, S.W., Trivedi, K.S., Vaidyanathan, K., Zeggert, W.P.: Proactive Management of Software Aging. IBM Journal of Research & Development 45(2), 311–332 (2001)
[5] Cinlar, E.: Introduction to Stochastic Processes. Prentice-Hall, New Jersey (1975)
[6] Dohi, T., Goseva-Popstojanova, K., Vaidyanathan, K., Trivedi, K.S., Osaki, S.: Software Rejuvenation: Modeling and Applications, Handbook of Reliability Engineering, ch.14, pp. 245–263. Springer, London (2006)
[7] Dohi, T., Goseva-Popstojanova, K., Trivedi, K.S.: Estimating software rejuvenation schedules in high-assurance systems. The Computer Journal 44, 473–482 (2001)
[8] Eto, H., Dohi, T.: Determining the Optimal Software Rejuvenation Schedule via Semi-Markov Decision Process. Journal of Computer Science 2(2), 528–535 (2006)
[9] Grotke, M., Trivedi, K.S.: Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. Computer 40(2), 107–109 (2007)
[10] Huang, Y., Kintala, C., Kolettis, N., Fulton, N.D.: Software rejuvenation: analysis, module and applications. In: Proceedings of 25th International Symposium on Fault Tolerant Computer Systems, pp. 381–390. IEEE CS Press, Los Alamitos (1995)
[11] Jiang, L., Xu, G.: Modeling and analysis of software aging and software failure, Journal of Systems and Software, Software Performance, 5th International Workshop on Software and Performance, vol. 80(4), pp. 590–595 (2007)
[12] Koutras, V.P., Platis, A.N.: Semi-Markov Performance Modeling of a Redundant System with Partial, Full and Failed Rejuvenation. International Journal of Critical Computer Based Systems, pp. 59—85. Inderscience Publishers (2009)
[13] Koutras, V.P., Platis, A.N.: Modeling Perfect and Minimal Rejuvenation for Client Server Systems with Heterogeneous Load. In: 14th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 95–103. IEEE Computer Society Press, Los Alamitos (2008)
[14] Koutras, V.P., Platis, A.N., Limnios, N.: Availability and Reliability Estimation for a System Undergoing Minimal, Perfect and Failed Rejuvenation. In: IEEE International Conference on Software Reliability Engineering Workshops, ISSRE (2008), doi:10.1109/ISSREW.2008.5355519
[15] Parnas, D.L.: Software aging. In: Proceedings of 16th International Conference on Software Engineering, pp. 279–287. ACM Press, New York (1994)
[16] Platis, A.N.: A generalized formulation for the performability indicator. Computers & Mathematics with Applications 51(2), 239–246 (2006)
[17] Platis, A.N., Drosakis, E.: Coverage modeling and optimal maintenance frequency of an automated restoration mechanism. IEEE Transactions on Reliability 58(3), 470–475 (2009)
[18] Rezaei, A., Sharifi, M.: Rejuvenating High Available Virtualized Systems. In: International Conference on Availability, Reliability and Security, pp. 289–294 (2010)

[19] Rinsaka, K., Dohi, T.: A Faster Estimation Algorithm for Periodic Preventive Rejuvenation Schedule Maximizing System Availability. In: Malek, M., Reitenspieß, M., van Moorsel, A. (eds.) ISAS 2007. LNCS, vol. 4526, pp. 94–109. Springer, Heidelberg (2007)
[20] Rinsaka, K., Dohi, T.: Behavioral analysis of fault-tolerant software system with rejuvenation. IEICE - Transactions on Information and Systems E88-D(12), 2681–2690 (2005)
[21] Sullivan, M., Chillarege, R.: Software Defects and Their Impact on System Availability—A Study of Field Failures in Operating Systems. In: Proceedings of the 21st IEEE International Symposium on Fault-Tolerant Computing, pp. 2–9 (1991)
[22] Tai, A., Tso, K.S., Sanders, W.H., Chau, S.N.: A performability-oriented software rejuvenation framework for distributed applications. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN-2005), pp. 570–579 (2005)
[23] Tai, A.T., Alkalai, L., Chau, C.N.: On board Preventive Maintenance for Long life Deep space Missions: A model based evaluation. In: Proceedings of 3rd IEEE International Computer Performance and Dependability Symposium, pp. 196–105 (1998)
[24] Thein, T., Chi, S.D., Park, J.S.: Availability Modeling and Analysis on Virtualized Clustering with Rejuvenation. International Journal of Computer Science and Information Security 8(9), 72–80 (2008)
[25] Trivedi, K.S.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 2nd edn. John Wiley & Sons, Chichester (2001)
[26] Xie, W., Yiguang, H., Trivedi, K.S.: Analysis of a two-level software rejuvenation policy. Reliability Engineering and System Safety 87(1), 13–22 (2005)

Towards a UML Profile for Maintenance Process and Reliability Analysis

Marcin Kowalski and Jan Magott

Wrocław University of Technology, Wybrzeze St.Wyspianskiego 27,
50-370 Wrocław, Poland
email: marcin.kowalski@pwr.wroc.pl, jan.magott@pwr.wroc.pl

Abstract. In the field of reliability analysis just few modeling languages became widely recognized by specialists. Being born out of their simplicity, Fault Trees continuously held appeal to both scientists and engineers, which brought them various syntactical extensions as well as analysis software. Although, Fault Trees have proved their usefulness by nontrivial technical problems, we notice their weakness in specification of maintenance processes, which are usually composed of actions taken in accordance with control- and dataflow. For this reason we try merging UML 2.0 Activity Diagrams with Probabilistic Fault Trees with Time Dependencies by expressing events as objects. That in fact requires redefinition of generalization and causal gates of the trees, though lays the foundations for the new language named Reliability-Enhanced Activity Diagram. By taking advantage of the UML 2.0 infrastructure, we design the language's profile and validate the approach against a model of a computer system's repair scheme.

1 Introduction

By firmly grasping the basic ideas of classical logic, the Fault Tree [6] formal language brought something of a novelty into dependability modeling. Doubtless, capturing the generalization relationship between events using gates resembling logical operators turned out to be a success story among engineers focusing on structures of systems being modeled. Therefore, various language extensions, such as: Repair Fault Trees, Dynamic Fault Trees or Probabilistic Fault Trees with Time Dependencies were explored to edge the formalism's way through a particular aspect of dependability modeling.

Factors that increased applicability of Fault Trees were the following papers: [5], where dynamic fault trees have been introduced and [3], where repair boxes have been defined. In the RELEX tool [18], dynamic gates are converted into Markov models. In [3], elements of Dynamic Fault Trees (DFT) and repair boxes are translated into such a subclass of colored Petri nets that is called stochastic well formed nets. The last ones are converted into Markov models. In paper [12], translation from dynamic fault trees into Bayesian networks has been presented. Formal tools: Markov models, Petri nets, Bayesian networks are not popular among engineers. However, descriptive power of the Repair Fault Trees, Dynamic

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 117–129.
springerlink.com

Fault Trees when such time dependencies like a sequence of time consuming activities or time redundancy have to be expressed is strictly limited. Therefore, Probabilistic Fault Trees with Time Dependencies have been introduced in [1].

The main challenge in the search for prominent extensions is to increase expressive power of the language in such a skillful manner that its intuition to engineers is left intact.

Up to now, process modeling in Fault Trees was confined to simple cases, e.g. repairs are represented in Repair Fault Trees [4]. Maintenance processes are much more complicated. They contain not only repairs, but also such activities as: testing, preventive maintenance, corrective maintenance that are important factors in maintenance optimization.

In paper [15], original twenty control flow-patterns plus identified twenty three new patterns relevant to control-flow perspective have been presented and formally expressed in Colored Petri nets. An evaluation obtained from detailed analysis of the control-flow patterns across fourteen commercial offerings including workflow systems, business process modeling languages and business process execution languages have been given. According to the evaluation UML Activity Diagrams 2.0 with BPMN and XPDL are in top three products of these fourteens.

Hence an idea presented in this paper to combine the UML Activity Diagrams (ADs) [13], a highly expressive and practically appreciated language, with probabilistic fault trees with time dependencies [1]. As a result, a wide range of behaviors such as: atomic actions, resource allocations, time-consuming activities as well as sequential and parallel activities with synchronization may be expressed in dependability models. Moreover, a profile for UML models built in IBM Rational Software Architect 8 [16] is provided, so that Reliability-Enhanced Activity Diagram (READ) models can be designed and validated. While building our profile, we incorporate the Modeling and Analysis of Real-Time and Embedded Systems Profile [17] to specify a timing model. The obtained modeling approach is tested against a computer system incorporating a repair facility allocation scheme.

A similar idea to combine UML ADs with Fault Trees to evaluate security has been a foundation of the paper [9]. In this profile, an emphasis is put on attack modeling while in our present paper we highlight failure/repair process consisting of control and data-ordered actions redefined in the 2.0 version of UML. In addition, we enhance ADs with Probabilistic Fault Trees with Time Dependencies whose expressive power is much greater than standard Fault Trees power. By incorporating the notion of time within gates and elaborating action semantics we deliver an executable UML profile, for which a dedicated simulator will be implemented.

READ models can be applied in agile software development process management [2]. In this process, testing is crucial activity. Model of this process with defect prediction can be useful in testing and debugging planning.

Structure of the paper is as follows. First, ADs in expressing maintenance process are presented. Then, Probabilistic Fault Trees with Time Dependencies adapted to ADs are outlined. In Section 4, the READ UML profile is shortly presented. Next, a case study of computer system with CPU, memory with redundancy and disc with redundancy is modeled in the READ UML profile. Finally, we conclude the research.

2 Activity Diagrams with READ Transitions in Maintenance Processes Modeling

In ADs, control-flow and data-flow can be represented. Sequential and parallel composition, decision, iteration, hierarchy (actions are nested in activities), different types of synchronization, multicasting, multireceiving, can be expressed. Detailed analysis of modeling power of ADs is given in [14], [15].

Now it will be shown how aspect of failure and repair processes can be described by ADs. Let us look at Fig. 1.

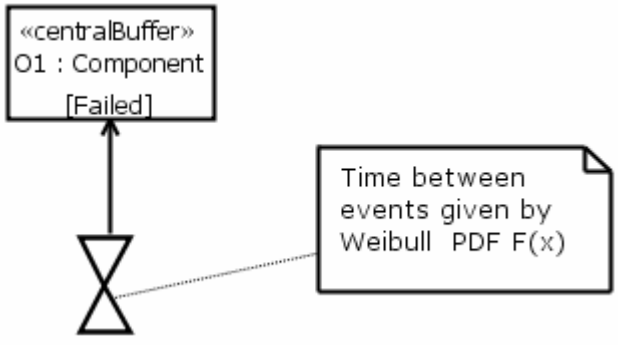


Fig. 1 Failure process of objects of a Component class

Hour-glass combined from two triangles is an Accept Time Event Action. It delivers time events. Length of time interval between two subsequent events is described by Weibull distribution with probability density function F(x). It expresses failure stream. Each time instant the event occurs, an object of Component class is located in central buffer. This object is in state "failed". Semantics of ADs is Petri net based one. Hence, an object located in central buffer is a token.

Objects of a class can be distinguished by a parameter that is a class attribute. The parameter can be an identifier. Let CPU be a class of central processing units. Its instances can be distinguished by their identifiers, e.g., CPU1, CPU2.

In fault trees there are events and gates. In the profile, events of fault trees are represented by objects that can be put into or drawn from central buffers.

Therefore, object located in central buffer can represent a component in different states or event. It is an advantage of generality level of object concept. It is clear when comparing with paper [8] where except typical fault tree constructs as events there are states.

Fork and Join nodes are elements of ADs. They usually are used in order to express a parallel composition. They both are represented graphically in the same way as a Petri net transition [10]. Additionally, the following constraint is imposed:

The edges coming into and out of a fork node must be either all object flows or all control flows. ([13], page. 387).

Therefore, in the READ UML profile, a Petri net transition symbol will be used in the meaning of the transition, neither Fork nor Join nodes. The ReadTransition can be used in modeling a repair process.

Let us consider Fig. 2.

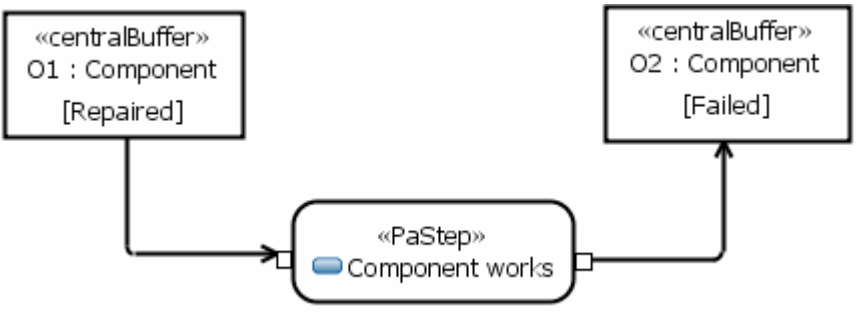


Fig. 2 The *Component works* activity with two central buffers

Length of time interval between start of work till *Component* failure is expressed by duration time of work, and it can be expressed by random variable (RV), constant, pair: minimal, maximal values or can be equal infinity (∞). In time instant when the *Component* fails, the token which represents repaired Component is removed from central buffer *Component* [*Repaired*], and the token which expresses failed *Component* is added to central buffer *Component* [*Failed*].

Fig. 3 illustrates repair facility allocation and start of repair.

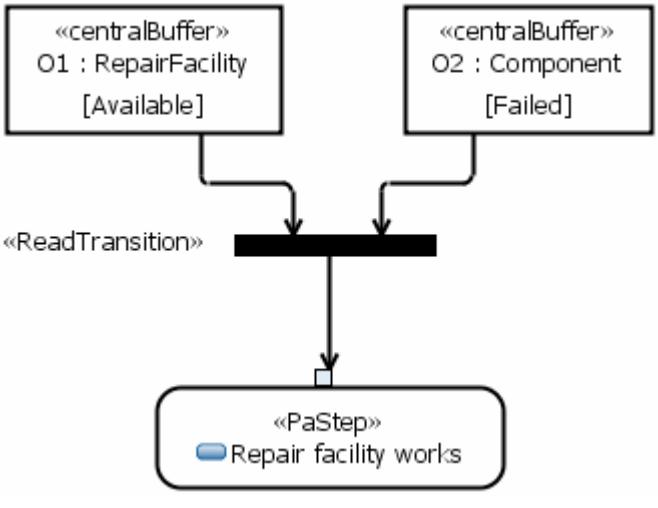


Fig. 3 Repair facility allocation and start of repair

The following figure shows end of repair and repair facility de-allocation.

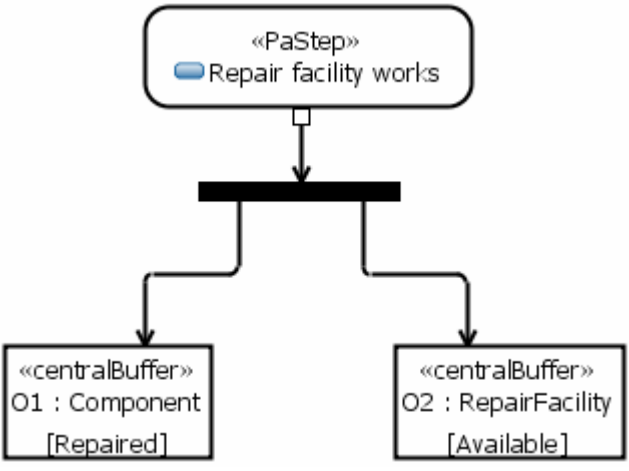


Fig. 4 End of repair and repair facility de-allocation

Since the UML State Machine specification also defines a notion of a Transition, which is a different construct than the one outlined above, we disambiguate the two by calling a Petri net based Transition a ReadTransition.

Furthermore, we introduce a concept of an inhibitor edge, which is an edge connecting an inhibiting central buffer with a ReadTransition element. The ReadTranstion cannot fire for the object with a parameter value p if there is an object with this parameter value in the inhibiting central buffer. Inhibitor edges are drawn as solid lines with an empty circle at their ends.

A collection of objects in particular states existing at the start of analysis is called Initial Object Set.

3 Probabilistic Fault Trees with Time Dependencies Adapted to Activity Diagrams

Probabilistic Fault Trees with Time Dependencies (PFTTD) [1] are combined from events, gates, and connections between them. The gates, similarly as for fault trees with time dependencies [7], [11], are divided into two categories: generalization and causal. Output event of generalization gate is a combination of input events. Causal gate is characterized by delay times between causes (input events) and effect (output event). Generalization gates are denoted by the 'G' symbol, whereas causal ones by 'C'.

Differences and similarities between logical gates of digital circuits and gates of the profile will be explained. The OR gate will be used as an illustration. Let us consider Fig. 5.

Fig. 5 Gates: G1 - the logical OR gate of digital circuits, G2 - the generalization OR gate of the profile, G3 - the causal OR gate of the profile

The logical OR gate of digital circuits (Fig. 5, G1) is described by the logical formula: $z = x \vee y$, where „$\vee$" is a symbol of the OR logical operator, x, y – input variables, z – an output variable. It is a formal model; however, in practice logical gates are also characterized by propagation time of input signals to output. Input, output signals represent input, output variables respectively.

A generalization OR gate (Fig. 5, G2) is defined in the following manner.

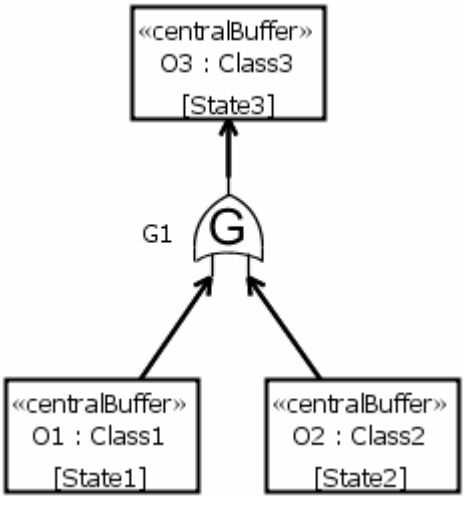


Fig. 6 The generalization OR gate

An object x occurs in a central buffer in time interval expressed by instant $\tau(xs)$ when it has been located into this buffer till instant $\tau(xe)$ when it has been removed from this buffer. In a particular case, an object occurring time in the buffer can be equal to zero, i.e., $\tau(xs) = \tau(xe)$.

Let $x(t)$ denote Boolean logical value of variable x in time instant t. Let T (F) denote logical value True (False).

Object object:class [state] occurs in its central buffer in time instant t iff Boolean variable object:class [state](t)=T.

Let $z = B(x_1,...,x_n)$ be Boolean function, where $x_1,...,x_n,z$ are Boolean variables.

Generalization gate of *B* function type is defined as:

$$z(t) = T \Leftrightarrow B(x_1(t),...,x_n(t)) = T .$$

Let us analyze generalization OR (GOR) gate in Fig. 6:

This gate behaves according to formal model of logical circuits OR gate, i.e., without propagation delays.

At some time instant, an object of Class3 in State3 is present in the output central buffer if there is either an object of Class1 in State1, or an object of Class2 in State2 present in at least one of the input central buffers.

In order to explain causal OR gate (COR, Fig. 5, G3), the following abbreviations are given:

x – an object of Class1 in State1,
y – an object of Class2 in State2,
z – an object of Class3 in State3,
occur(x) – object x has occurred in a central buffer Class1 [State1],
d1 (*d2*) - random variable (RV) that represents time delay between the instance when an object either x or y was located in an input central buffer and the instance when the object z was located in the output central buffer,
R(X) - a realization of random variable X, i.e., a value generated according to the distribution of the X.

A causal OR (COR) gate is given by a following expression:

$$occur(z) \Rightarrow$$
$$(((occur(x) \wedge \tau(zs) = \tau(xs) + R(d1)) \vee (occur(y) \wedge \tau(zs) = \tau(ys) + R(d2)))$$

Meaning of the above formulae is as follows: if the object z occurred in output central buffer then the object x or y had occurred in input central buffer. The object z was located in the output central buffer in instant *R(d1)* or *R(d2)*, respectively, with reference to the instant when the object *x* or *y* was located in the input central buffer.

Let *n-occur(x,t)* denote the number of objects *x* of Class in State that are occurring in the central buffer Class [State] in time instant *t*.

Generalization voting gate G Vote with threshold *k* with input central buffers Class1 [State1], Class2 [State2], and output central buffer Class3 [State3] is defined as follows:

$$occur(z,t) \Rightarrow n\_occur(x,t) + n\_occur(y,t) \geq k$$

where:

x – an object of Class1 in State1,
y – an object of Class2 in State2,
z – an object of Class3 in State3.

The meaning of the above formula is as follows: if an object z is occurring in output central buffer at time instant t then at least k objects are occurring in input central buffers at this instant.

Example of this gate type is G2 in Fig. 10. The threshold of the gate is 2. Examples of other PFTTD gates are given in [1].

4 The READ UML Profile

Having outlined the new language, we extend UML using IBM Rational Software Architect 8 producing a new UML profile depicted in Fig. 7. We plug into the host language through *ActivityEdge* and *ActivityNode* constructs, which lay the graph-wise foundations of an activity diagram. To this end we use the Extension associations drawn as solid lines with black, filled arrows connecting ReadNode with ActivityNode and InhibitorEdge with ActivityEdge. The ReadNode abstract stereotype underlies the profile core, because it is the most general construct specialized by ReadTransition, CGATE and GGATE. From the two last stereotypes concrete gates are derived. Currently AND, OR, PAND, CXOR and VOTE in both generalization and causal forms are supported.

By extending ActivityEdge we advocate for an inhibitor edge, being a specialized edge from a central buffer to a transition.

4.1 Timing Model

We recognize the effort to deliver a UML timing model made when designing the Marte profile. In particular, we take advantage of the PaStep stereotype accompanied by its non-functional properties (NFP) to specify three types of timing requirements:

duration of an activity
lifetime of an object contained in a CentralBuffer node
delay of an object state change introduced by causal gate.

Hence, in order to define time to failure of a processor we assign the PaStep stereotype to the ‘CPU running’ action (Fig. 10) and define an NFP_Duration object, referenced by the ‘execTime’ stereotype attribute, as follows:

('assm','dist',('exp',1000000),'min')).

According to the Marte specification, the time to failure is assumed (*assm*) to be a random variable (*dist*) with an exponential distribution (*exp*). The mean time to failure is *1000000* minutes (*min*).

Following the above reasoning a timing model may be specified in the case of CentralBuffer objects. If lifetime is infinite, as for every CentralObject in Fig. 10, we refrain from applying the PaStep stereotype at all.

In the case of causal gates, different timing model is applied depending on the input node effectively activating the gate. Thus, instead of annotating the CGATE with PaStep, we provide a Delay class, whose objects are contained in a particular causal gate in the respective cardinality.

4.2 Elaborating Syntactical Validity

For many READ constructs we define OCL expressions pinpointing syntactical rules imposed on models. Some of these are shown in Fig 7. For example:

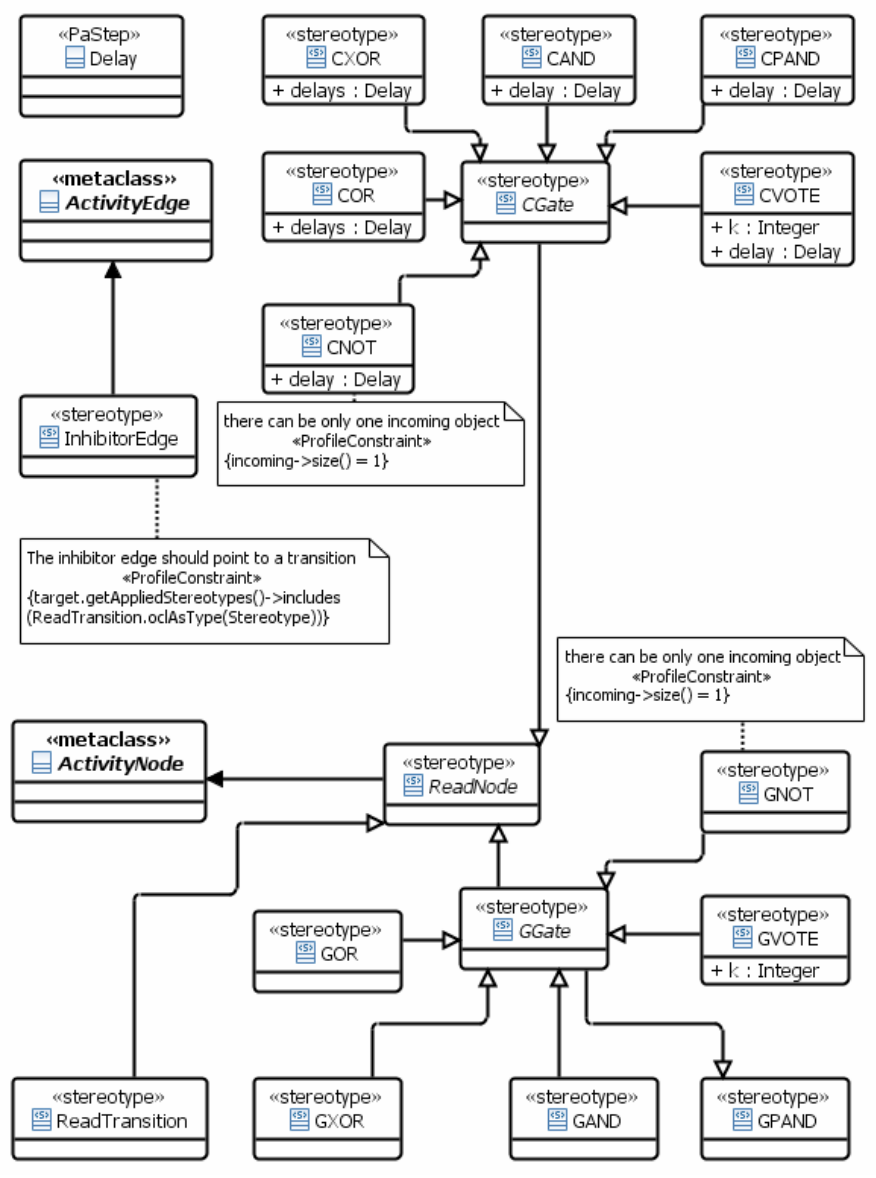


Fig. 7 The READ UML profile

CNOT and GNOT gates must have only one node at their incoming edges any InhibitorEdge construct must end with an inhibited ReadTransition element a number of Delay objects in the case of COR and CXOR gates must be equal to their number of incoming edges (not shown).

5 Case Study

In this section a computer system consisting of several components and repair facilities will be investigated by means of READ (Fig. 8). For the system to run properly, one CPU, one disc and one memory unit must be working properly, i.e. be in the "Running" state. However, to advocate for reliability, hot spares of disc and memory units were introduced.

When a component fails, it waits for an 'Available' repair facility and subsequently undergoes repair. Contrary, a repair facility is 'Busy' when servicing another component being in the 'UnderRepair' state.

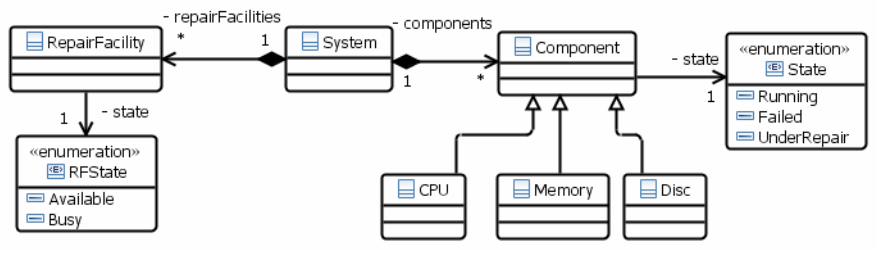


Fig. 8 Classes and enumerations modeling the case study system

All in all, in the case study maintenance process five components and two repair facilities comprise the system (Fig. 9). In the READ method, we consider UML Object Diagram to be defining the Initial Object Set.

To group actions, gates and objects, the case study model (Fig. 10) has been divided into three vertical partitions, those being Regular service, Repair and

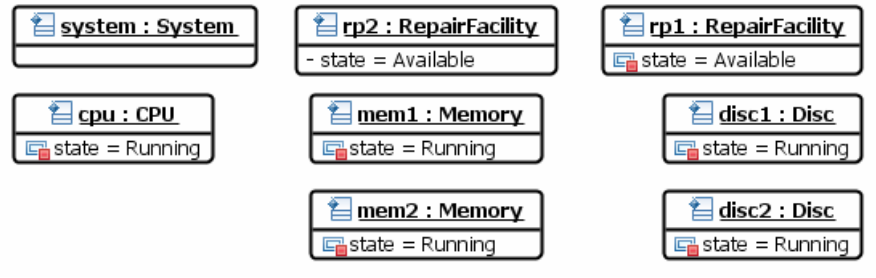


Fig. 9 Initial Object Set

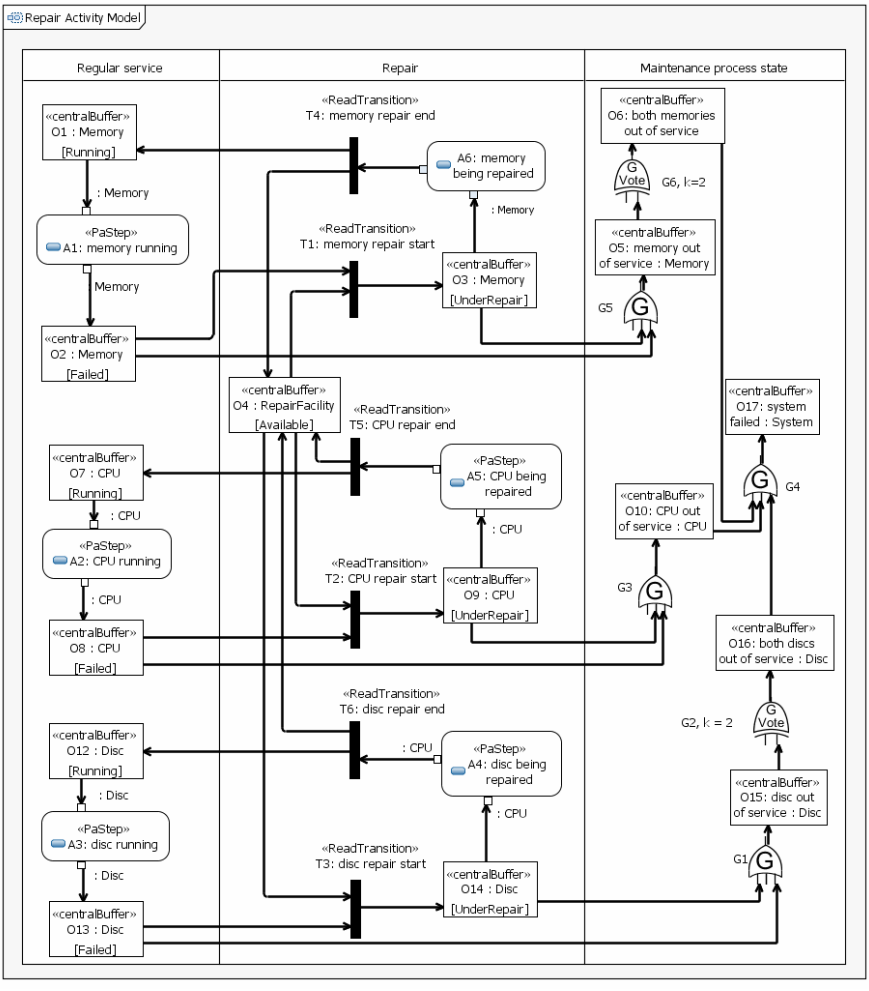


Fig. 10 The case study model

Maintenance process state. Objects flow horizontally between them as system components fail and repair. Let us analyze the top-most model section: O1, A1, O2, T1, O4, O3, G5, O5, G6, O6, A6 and T4 which apply to memories. The middle and bottom section work alike.

Objects from Fig. 9 fill in O1 and O4 central buffers of the top section at the initial analysis moment causing the A1 action to start two times in parallel (one for the "mem1" and the other for the "mem2" object). This action models correct memory operation, therefore its timing model defines a proper random variable. After A1 has finished for some memory unit (the first failed memory), the action removes the respective object from O1 and puts it into O2 changing its state to "Failed". Next, if a repair facility is available, the T1 transition realizes the

allocation scheme described in Section 1. As a result, the memory object is moved to the O3 buffer and its repair is initialized.

If the component is failed, or it is under repair, the G5 GOR gate puts an object into O5 buffer denoting that the component is out of service. If the second memory fails, O6 occurs through G6, and the system is failed (O17) on the virtue of the G4 GOR gate.

The A6 action lasts as long as the component is repaired. Next, the repair facility is released (T4) and the repaired memory is restored into the "Running" state (O1). This causes a change in the O5 buffer effectively restoring the whole system by removing objects from O6 and O17.

6 Conclusions

Out of many system modeling languages, just a few became practically appreciated owing to their intuitiveness and expressive power. In this paper we managed to combine two of them, those being fault trees and activity diagrams, and eventually came up with a new modeling language offering object oriented approach while specifying event composition, event causality and timing dependencies between events in the system being modeled. Apart from that, we introduced a Petri net based transition into the language substantially increasing its modeling power and enabling development of models, whose specification in fault trees was up to now virtually impossible.

As a proof of concept we presented one of those models, namely specification of a repair process applied to a computer system. That involved provision of a UML profile for the IBM Rational Software Architect platform.

We also seized the opportunity of reusing diagrams other than activity in order to build models. For example, by means of class diagrams we capture system's state space, whereas object diagrams allow us to describe the initial system state.

Next, we want to apply the READ technique to a real-life system and build a simulator to investigate the system's reliability.

References

[1] Babczyński, T., Łukowicz, M., Magott, J.: Time coordination of distance protections using probabilistic fault trees with time dependencies. IEEE Transaction on Power Delivery 25(3), 1402–1409 (2010)

[2] Bessam, A., Kimour, M.T., Melit, A.: Separating users' views in a development process for agile methods. In: Proc. International Conference on Dependability of Computer Systems, DepCoS - RELCOMEX 2009, pp. 61–68 (2009)

[3] Bobbio, A., Codetta, D.: Parametric fault trees with dynamic gates and repair boxes. In: Proc. Annual Symposium on Reliability and Maintainability, pp. 459–465 (2004)

[4] Codetta, D., Franceschinis, G., Iacono, M., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: Proc. Proceedings of the 2004 International Conference on Dependable Systems and Networks, DSN 2004 (2004)

[5] Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. 41(3), 363–367 (1992)
[6] Fault Tree Analysis (FTA), International Technical Commission, Publication 1025 (1990)
[7] Górski, J., Magott, J., Wardzinski, A.: Modelling fault trees using Petri nets. In: in Proc. SAFECOMP 1995, Belgirate, Italy. LNCS. Springer, Heidelberg (1995)
[8] Grunske, L., Kaiser, B., Papadopoulos, Y.: Model-Driven Safety Evaluation with State-Event-Based Component Failure Annotations. In: Heineman, G.T., Crnković, I., Schmidt, H.W., Stafford, J.A., Ren, X.-M., Wallnau, K. (eds.) CBSE 2005. LNCS, vol. 3489, pp. 33–48. Springer, Heidelberg (2005)
[9] Houmb, S.H., Hansen, K.K.: Towards a UML profile for security assessment. In: Proc. UML 2003, Workshop on Critical Computer Systems Development with UML, pp. 815-829 (2003)
[10] ISO/IEC 15909-1, High-level Petri nets: Concepts, definitions and graphical notation (2004)
[11] Magott, J., Skrobanek, P.: Method of time Petri net analysis for analysis of fault trees with time dependencies. In: IEE Proceedings - Computers and Digital Techniques, vol. 149(6), pp. 257–271 (2002)
[12] Montani, S., Portinale, L., Bobbio, A., Codetta-Raiteri, D.: ADYBAN: a tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. Reliability Engineering and System Safety 93(7), 922–932 (2008)
[13] OMG, Unified Modeling Language (OMG UML), Superstructure Version 2.3 (2010)
[14] Russel, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: On the suitability of UML 2.0 activity diagrams for business process modeling (2006)
[15] Russel, N., ter Hofstede, A.M.H., van der Aalst, W.M.P., Mulyar, N.: Workflow control-flow patterns, A revised view (2006)
[16] IBM, Developer Works on IBM Rational Software Architect (2011), https://www.ibm.com/developerworks/rational/
[17] OMG, The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems (2010), http://www.omg.org/omgmarte/Specification.htm
[18] RELEX, Resources on Fault Trees (2010), http://www.relex.com/resources/art

Conjoining Fault Trees with Petri Nets to Model Repair Policies

Marcin Kowalski and Jan Magott

Wrocław University of Technology, Wybrzeze St.Wyspianskiego 27,
50-370 Wrocław, Poland
email: marcin.kowalski@pwr.wroc.pl, jan.magott@pwr.wroc.pl

Abstract. Right from the beginning, the fault tree language gained great acceptance in reliability modeling, because it bears a striking resemblance to the operators found in the classical logic. Therefore, only by using 'AND' as well as 'OR' gates a number of system failures can be expressed even by engineers not related to reliability analysis. Doubtless, intuition accompanying fault tree models is their greatest merit. Therefore each attempt to increase their expressive power by introducing a set of very specific model extensions strives to retain the intuitiveness. The most remarkable extensions are dynamic gates and repair boxes. However, these extensions are strictly limited to expressing time dependencies like a sequence of time consuming activities or time redundancy. From this viewpoint, fault trees and Petri nets complement one another. The latter offer huge modeling power comparative to Turing machines, but their models turned out to be obscure to engineers. Hence, this severe limitation hampers widespread popularity of Petri Nets. By analyzing the constraints of the two languages, we come up with a new modeling technique blending fault trees with Petri nets. We extend the expressive power of fault trees by adding Petri net immediate transitions. The obtained fault graphs with time dependencies are investigated by modeling several repair policies on some exemplary computer system. Availability calculations of the system are possible owing to a dedicated tool.

1 Introduction

Standard fault trees [6] have been used in reliability and safety analysis for fifty years. Right from the beginning, the fault tree language gained great acceptance in reliability modeling, because it bears a striking resemblance to the operators found in the classical logic. Therefore, only by using 'AND' as well as 'OR' gates a number of system failures can be expressed even by engineers not related to reliability analysis. Doubtless, intuition accompanying fault tree models is their greatest merit. Therefore each attempt to increase their expressive power by introducing a set of very specific model extensions strives to retain the intuitiveness.

They do not have great power of expressing the real systems. Factors that increased applicability of Fault Trees were the following papers: [5], where dynamic

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 131–143.
springerlink.com

fault trees have been introduced and [3], where repair boxes have been defined. In tool RELEX [14], dynamic gates are converted into Markov models. In [3], elements of Dynamic Fault Trees (DFT) and repair boxes are translated into such a subclass of colored Petri nets that is called stochastic well formed nets. The last ones are converted into Markov models. In paper [13], translation from dynamic fault trees into Bayesian networks has been presented. Formal tools: Markov models, Petri nets [9], Bayesian networks are not popular among engineers. However, descriptive power of the above extensions when such time dependencies like a sequence of time consuming activities or time redundancy have to be expressed is strictly limited.

From this viewpoint, fault trees and Petri nets complement one another. The latter offer huge modeling power comparative to Turing machines, but their models turned out to be obscure to engineers. Hence, this severe limitation hampers widespread popularity of Petri Nets.

By analyzing the constraints of the two languages, we come up with a new modeling technique blending fault trees with Petri nets. We extend the expressive power of fault trees by adding Petri net immediate transitions. The obtained fault graphs with time dependencies are an extension of probabilistic fault trees with time dependencies [2]. The fault graphs are investigated by modeling several repair policies on some exemplary computer system. Availability calculations of the system are possible owing to a dedicated tool.

Structure of the paper is as follows. First, probabilistic fault trees with time dependencies are presented. Next, fault graphs with time dependencies are outlined. In the third section we evaluate various repair policy models expressed as fault graphs. We conclude the chapter in the fifth section.

2 Probabilistic Fault Trees with Time Dependencies

Probabilistic fault trees with time dependencies (PFTTD) are combined from events, gates, and connections between them. The gates, similarly as for fault trees with time dependencies [7], [8], [11], [12], are divided into two categories: generalization and causal. Output event of generalization gate is a combination of input events. Causal gate is characterized by delay times between causes (input events) and effect (output event). Generalization gates are denoted by the ‘G’ symbol, while causal ones by ‘C’.

Differences and similarities between logical gates of digital circuits and gates of PFTTDs will be explained. The AND gate will be used as an illustration. Let us consider Fig. 1.

Any logical AND gate of digital circuits (Fig. 1 a)) is described by logical formula: $z = x \wedge y$, where „$\wedge$” is a symbol of AND logical operator, x, y – input variables, z – output variable. It is a formal model. Real-life logical gates are

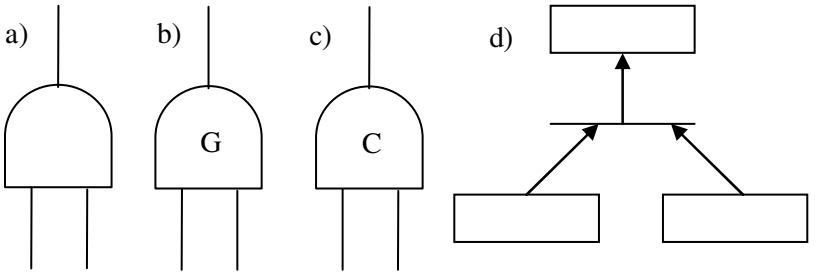


Fig. 1 Gates: a) logical AND gate of digital circuits, b) generalization AND gate of PFTTD, c) causal AND gate of PFTTD, d) transition with two input events and one output event.

characterized by a propagation time stretch of input signal to output. Input, output signals represent input, output variables respectively.

For the logical AND gate of digital circuits from Fig. 2 a) the formal model of time characteristics is in Fig. 2 b), while real-life time characteristics is in Fig 2 c).

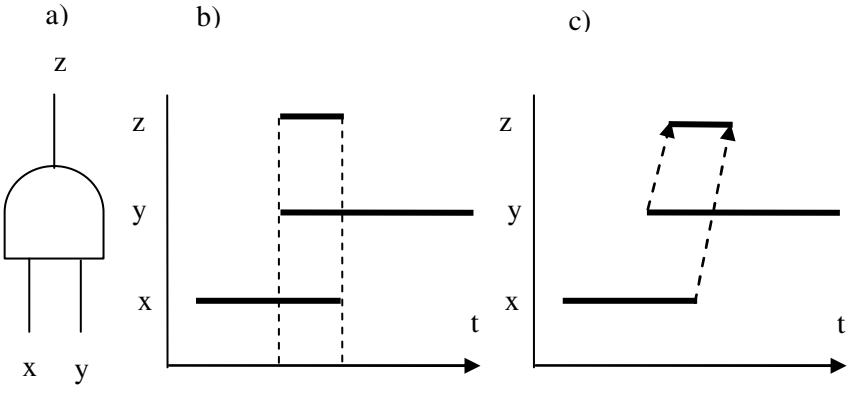


Fig. 2 a) the logical AND gate of digital circuits, b) its formal model of time zcharacteristics, c) its real-life time characteristics

A PFTTD event x occurs in a time interval expressed by start instant $\tau(xs)$ and end instant $\tau(xe)$ of this event. In a particular case, event duration time is equal to zero, i.e.: $\tau(xs)=\tau(xe)$.

Let $x(t)$ denote Boolean logical value of a variable x at time instant t. Let T (F) denote logical value *True* (*False*).

Event x occurs at time instant t iff Boolean variable $x(t)=T$.

Let $z=B(x_1,...,x_n)$ be Boolean function, where $x_1,...,x_n,z$ are Boolean variables.

A generalization gate of *B* function type is defined as:

$$z(t) = T \Leftrightarrow B(x_1(t),...,x_n(t)) = T .$$

Hence, for the generalization AND (GAND) gate in Fig. 1 b) the following holds:

$$occur(z) \Rightarrow occur(x) \wedge occur(y) \wedge overlap(x, y) \wedge$$
$$\max(\tau(xs), \tau(ys)) = \tau(zs) \wedge \min(\tau(xe), \tau(ye)) = \tau(ze)$$

where *occur*(*z*) is the logical formula with meaning: the output event (effect) *z* has occurred.

The meaning of the above formula is the following: the output event occurs only if both the input events have occurred and overlapped. The output event lasts as long as the input events co-occur. Therefore, its start is equal to the start of event which has started later. Its end is an end of the event that has been ended earlier.

Time characteristics of the GAND gate are the same as formal model for the logical AND gate of digital circuits (Fig. 2 b)).

Let *R*(*X*) be a realization of a random variable *X*, i.e., a value generated according to the distribution of *X*.

Causal AND (CAND) gate (Fig. 1 c)) is given by the expression:

$$occur(z) \Rightarrow ((occur(x) \wedge occur(y) \wedge$$
$$(\tau(xs) \le \tau(ys) \Rightarrow \tau(zs) = \tau(ys) + R(d)) \wedge$$
$$(\tau(ys) \le \tau(xs) \Rightarrow \tau(zs) = \tau(xs) + R(d)))$$

where: *occur*(*z*) is the logical formula with meaning: the output event (effect) *z* occurred, *d* - random variable (RV) that represents time delay between the occurrence (start) of the later cause *x* or *y* and the effect *z*.

Output event of this gate occurs (starts) at time instant *R(d)* with respect to instant when later input event occurs.

Time characteristics of CAND gate are depicted in Fig. 3.

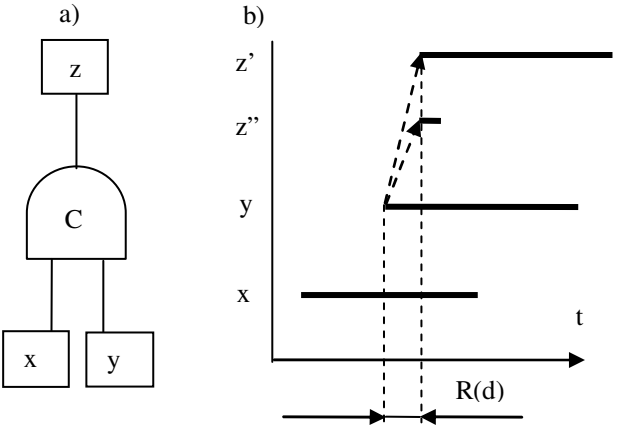


Fig. 3 a) A causal AND gate, b) its time characteristics

Let us analyze an example when the events from Fig. 3 are as follows:

x – gas burner failure, y – opened gas valve, z' – destroyed building, z" – explosion. If the gas valve is open and the gas burner does not supply fire, the gas concentration increases, and an explosion occurs at time instant *R(d)* with respect to the instant gas valve became opened. The output event can be also a long-lasting time event, e.g. the destroyed building.

3 Fault Graphs with Time Dependencies

An event *x* that is an element of PFTTD is illustrated by a rectangle with its duration time *t* (Fig. 4 a)). The duration time can be expressed by a random variable (RV), constant or can be equal to infinity (∞).

Events that are elements of fault graphs with time dependencies (FGTDs) are of two types; simple and parametric. Simple ones are described in the same way as in PFTTD. Apart from its duration time, a parametric event is characterized by a value $p \in P$, where *P* is a set of event *y* parameter values (Fig. 4 b)). The parameter can be, e.g., an event identifier.

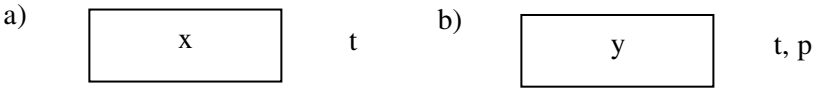


Fig. 4 Events: a) simple, b) parametric

FGTDs contain PFTTD gates that are called simple ones. Additionally, FGTDs include parametric gates. A parametric gate has the following property: all its input and output events are parametric with equal parameter values sets. Graphical representation of FGTD parametric gate of a type is the same as of PFTTD gate of the same type. Output event (effect) has the same parameter value as input events (causes) parameter value.

Additionally when comparing with PFTTDs, FGTDs contain a delay gate. This gate is illustrated by an hour-glass symbol with two triangles connected by nodes as gate G4 in Fig. 8. The time parameter of the gate expresses delay between start of input event and start of output event. It can be a random variable. For G4, this delay is described by an exponential RV with the mean value $2 \cdot 10^6$ h.

Causal NOT (CNOT) with input event x and output event y, and delay time expressed by random variable *d* is defined as follows:

$$occur(x) \Rightarrow (\forall \tau \in R_+)(\tau(xs) + R(d) < \tau \Rightarrow \neg occur(y, \tau)).$$

The meaning of the above formula is as follows: if input event has occurred then output event cannot occur after time instant $\tau(xs) + R(d)$.

Delay time for gate G5 in Fig. 8 is 0.

Time characteristics of this gate are in Fig. 5.

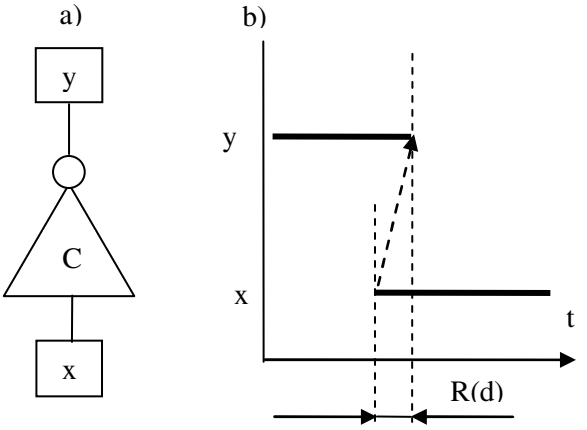


Fig. 5 a) A causal NOT gate, b) its time characteristics

In comparison with PFTTD, in FGTD there are transitions inspired by Petri nets. Two types of transitions are: simple and parametric. Simple transitions have the following property: all their input and output events are simple ones.

Firing of the transition from Fig.1d causes completion of its input events and occurrence (start) of the output event. Transitions have been introduced in order to represent repetitive processes.

A simple transition with an inhibitor event *v* is given in Fig. 6.

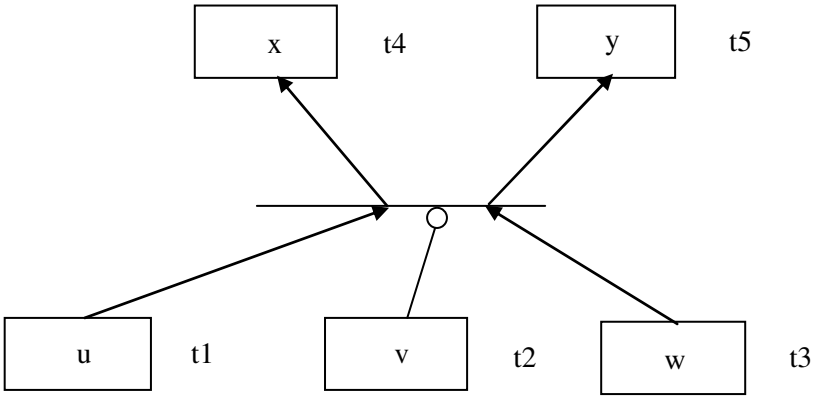


Fig. 6 Simple transition with three input events *u*, *v*, *w*, including inhibitor one *v*, and two output events *x*, *y*

The arc connecting the transition with the event *v* has a circle instead of arrow at its end. The transition can be fired at a time instant when events *u*, *w* occur but the inhibitor event *v* does not occur. Inhibitor events have been included in the FGTD language in order to increase expressive power, because they enable testing for zero.

A parametric transition has the property: at least one of its input or output events is the parametric one (see Fig. 7).

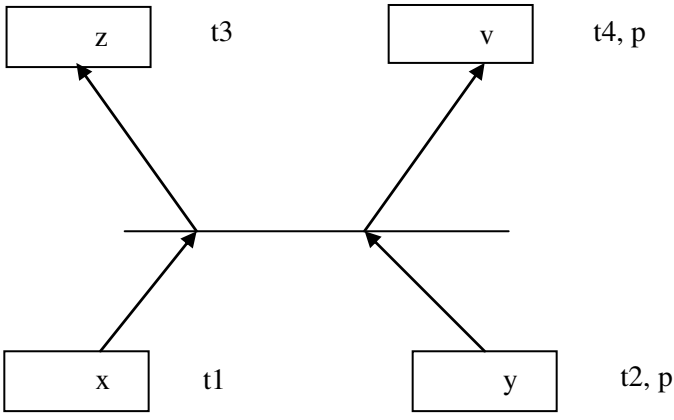


Fig. 7 A parametric transition with simple events *x*, *z* and parametric events *y*, *v*

When a parametric transition is fired, the following condition has to be satisfied: all parametric events that participate in firing must have the same parameter value.

Let us consider the following case. In the same time instant, a transition is enabled to be fired, and all input events of a causal gate with zero delay time occur. Hence, there is a conflict: what should be the first: transition firing or output event of the causal gate occurring. In this case, the transition firing will occur as the first one.

Let *n_occur(x,t)* denote the number of events *x* that are occurring at time instant *t*.

Generalization voting gate G Vote with threshold *k* with input events *x*, *y*, and output event *z* is defined as follows:

$$occur(z,t) \Rightarrow n\_occur(x,t) + n\_occur(y,t) \geq k\,.$$

The meaning of the above formula is as follows: if output event is occurring at time instant *t* then at least *k* input events are occurring at this instant.

Example of this gate type is G10 in Fig. 8. The threshold of the gate is 2.

Let the EX symbol denote event. In case of events with identifiers: the EX{a, b} notation denotes that events EX with identifiers a and b occur. In case of events without identifiers, the EX{n} notations denotes that event EX occurs n times.

4 Applying Fault Graphs with Time Dependencies to Selected Repair Policies

In this section a computer system with redundancy undergoing several repair policies will be investigated. The computer system consists of five components: single processor, two discs and two memory units, which fail and might be repaired in

accordance with rates presented in Table 1. In order for the system to run the processor and at least single memory and disc must be working.

Table 1

Component	Failure rate [1/h]	Repair rate [1/h]
CPU	$5*10^{-7}$	10^{-2}
Disks	$8*10^{-5}$	10^{-2}
Memories	$3*10^{-8}$	10^{-2}

There is an initialization time (cost) required to start every repair policy. For some policies there is a fixed number of repair facilities working concurrently to repair failed components. A single repair facility can repair at maximum a single component at some time moment.

Following repair policies will be applied to the system:

Global Repair Policy – GRT (T)– a repair policy is started when the systems fails and lasts a fixed period of time (T) after which all components are restored, there are no repair facilities.

SRT-F (N) – a repair process is started after the system fails and failed components are restored in a random order by a fixed number (N) of repair facilities, restored components are put into action after a whole repair process has ended.

SRT-I - we emulate infinite number of repair facilities by assigning N to 5 in SRT-F (since there may be at most 5 failed components).

Synchronized Start, Immediate Restoration Policy – SSIR - a repair process is started only after a whole system fails, but repaired components are immediately put into action, there are 5 repair facilities.

Immediate Start, Immediate Restoration – ISIR - a repair process is started immediately after any component has failed, repaired components are immediately put into action, there are 5 repair facilities.

Various policy configurations were analyzed by means of a generic Fault Graph simulator. These were: GRT(10), GRT(250), SRT-F(1), SRT-F(2), SRT-I, SSIR and ISIR. Apart from SSIR and ISIR, availability results can be compared (Table B) with [4].

For the sake of clarity, numeration of gates, events and transitions is consistent among models of different policies.

4.1 The GRT Repair Policy

The system is failed at some point in time if the E9 event occurs (Fig. 8). This is the case when (by virtue of the G13 GOR gate) either: E4 or E7 or E8 occurs, which rises from the fact that all three component types may cause the system to break down. Hence, three event columns along with adjacent gates and transitions comprise the model: E1, E4 as the first, E2, E5, E7 as the second and E3, E6, E8 as the last one. Since they look alike, they will be discussed simultaneously.

The E4 (E7, E8) occurs if all components of the respective type are failed. To model that, we take advantage of the G10 (G11) voting gate, which acts when a particular threshold (written after a semicolon in the gate's label) of events at its input is reached. Since there is only one processor, there is no need for a dedicated voting gate and E4 plays a similar role to E7 and E8. Taken that into account, to specify maintenance state properly, the E4 (E5, E6) event must be started (G4, G6, G8) and finished (G5, G7, G9) at the policy specified moments.

The E1 {0}, E2 {0,1}, E3 {0,1} belong to the initial event set, so they occur at the start of analysis. Symbols 0 and 1 denote identifies of CPU, memories, and discs. Hence, the G4 (G6, G8) time delay is initialized, i.e. E4 (E5, E6) event with respective identifier will occur at some time instant defined by the random variable denoting time to failure of a particular component type. Since duration of E1 (E2, E3) is "0.0", it stops immediately. When some E4 (E5, E6) occurs, effectively starting the E9 event, a new repair process is started. In this case, a total renewal occurs after a time stretch defined by the G14 gate. Whereas in Fig. 8 this time is equal to 10 minutes, to remain consistent with work [4] simulation was also run with a 250 minute delay (Table 2). Before E0 ends, it may cause ——E1 (E2, E3) to start by means of G1 (G2, G3) gate. If some component has been broken, it is now renewed. Since the

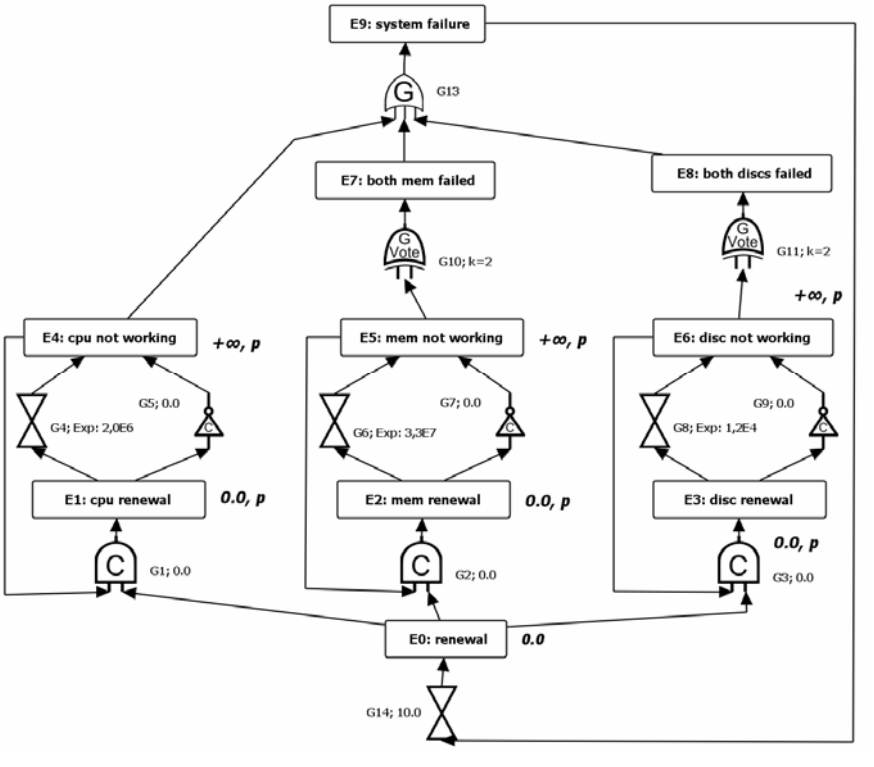


Fig. 8 The global repair policy model

component is working, E4 (E5, E6) is stopped by G5 (G6, G7). That in turn causes E8, E10 and E12 to finish, therefore putting E9 out as well. The system is restored and new failures have been scheduled by G4, G6 and G8.

Since all failed components are repaired concurrently and there is no challenge among them, we managed to model the policy without resorting to the transition extension.

4.2 The SRT-F/SRT-I Repair Policies

In the next model (Fig. 9) the three columns comprise are: E1, E4, E8, E11, E14, E18 as the first, E2, E5, E9, E12, E15, E19 as the second and E3, E6, E10, E13, E16, E20 as the last one.

The E1 {0}, E2 {0,1}, E3 {0,1} events along with the E18 {0}, E19 {0,1}, E20 {0,1} belong to the initial event set. At the beginning, we follow the reasoning for the GRT policy. The G4 (G6, G8) time delay is initialized, i.e. E4 (E5, E6) event with respective identifier will occur at some time moment in the future. In the due time, when the component is broken down, E18 (E19, E20) is stopped by virtue of the G18 (G19, G20) CNOT gate.

A total number of repair facilities (N) in a particular policy is denoted by E17 {N} belonging to the initial event set. For the SRT-I repair policy, N=5.

When E4 (E5, E6) occurs, G12 (G13, G14) causes waiting for (E8, E9, E10): repair start (E7) and repair facility (E17). A new repair process is started only after the whole system breaks down (the E23 event occurs), but it is actually running when the initialization time (G11, 10 minutes) passes and E7 starts. The E7 event will remain active as long as there is any component requiring repair. Should a new component fail while some repair process is already running, its repair is incorporated instead of being postponed until the subsequent repair process. Therefore, the initialization time has to pass once only.

When a repair process is started, repair facilities are allocated (locked and released) on demand in a following manner. The T1 (T2, T3) transition locks a facility by ending E17 but starting E11 (E12, E13). At that point, the processor (memory, disc) can be repaired, which takes time defined by the random variable attributing the G15 (G16, G17) gate. After the repair has finished, the repair facility is released to the pool (i.e. E17 is started) by the T4 (T5, T6) transition. The E18 (E19, E20) event is started anew. All in all, the T1-T3 transitions lock atomically repair facilities (ending E17), whereas by means of the T4-T6 transition, the repair facility is released (E17 started).

Notably, repaired components are not put into action immediately after repair. They can be repaired, though not working (e.g. E4 and E18 occurring at the same time). The reason for this is the policy stating that a repair process must end before putting any repaired component into action. It happens by means of the G23 gate asserting that all 5 components are not failed before triggering renewal (E0). Then, all the failed components are renewed instantly through the G1, G2, G3 gates starting E1, E2 and E3, which in turn finish E4, E5 and E6 through CNOT gates, because the components are no longer failed. Finally, the repair process is stopped by the G10 gate.

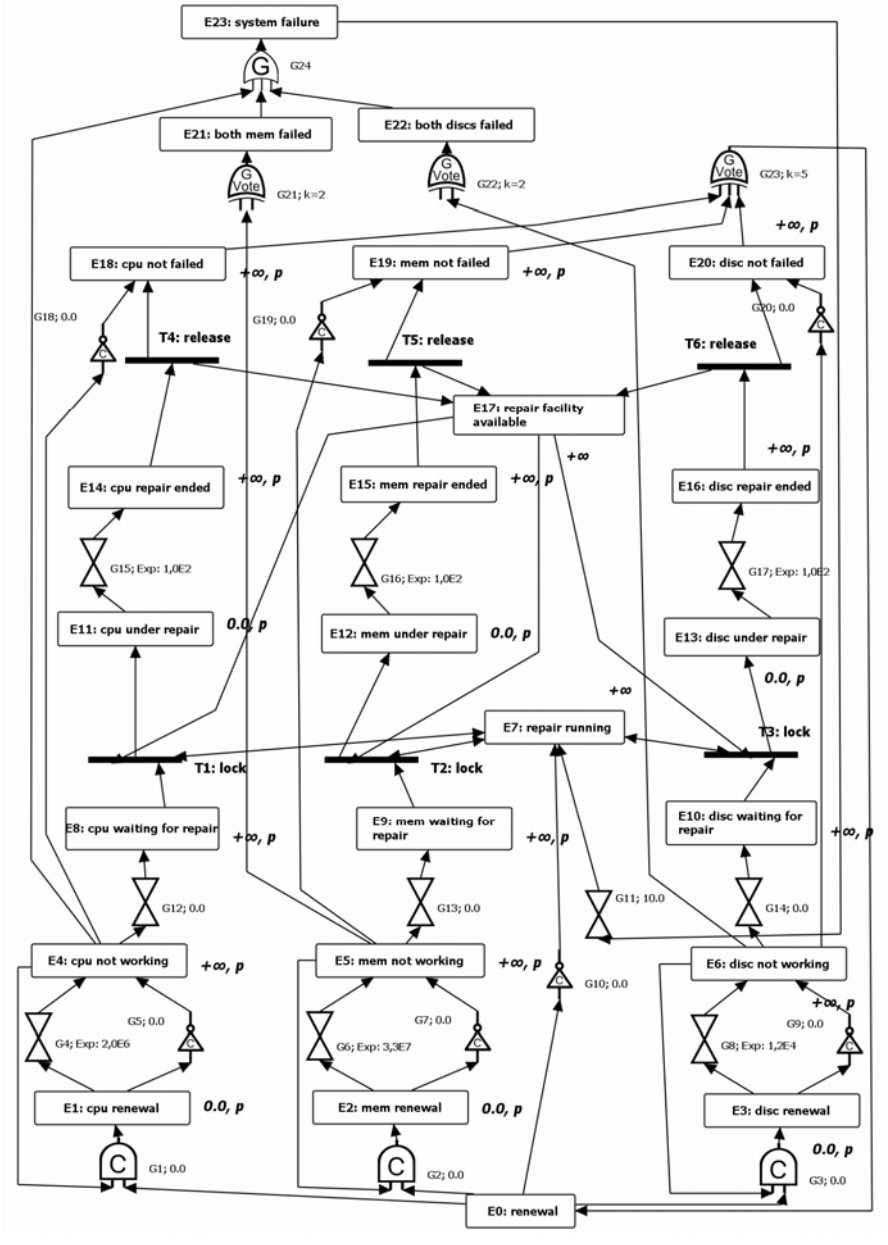


Fig. 9 SRT-F/SRT-I policy family model

Table 2 Unavailability results under various repair policies

Policy	Unavailability from [4]	Unavailability using Fault Graphs	Error [%]
GRT (T=10)	$5{,}37*10^{-4}$	$5{,}37*10^{-4}$	0
SRT-I	$8{,}517*10^{-3}$	$8{,}517*10^{-3}$	0
SRT-F (1 resource)	$1{,}11147*10^{-2}$	$1{,}11120*10^{-2}$	0,03
SRT-F (> 2 resources)	$8{,}517*10^{-3}$	$8{,}517*10^{-3}$	0
GRT (T=250)	$1{,}3253*10^{-2}$	$1{,}3241*10^{-2}$	0,1

5 Final Remarks

Using the FGTD language we managed to capture all the repair policies described in [4] resorting to neither highly specialized modelling constructs, as was the case with repairable fault trees using repair boxes, nor an obscure internal language like Petri nets. Although, to remain consistent with [4] we took a simple computer system as a case study, our approach could scale well into systems with sophisticated timing requirements like the ones discussed in [2] or [10], whose modelling in standard fault trees is merely out of question. We have therefore filled the gap between modelling systems and modelling their repair policies, which was the goal we set ourselves in the first place when failing to apply repairable fault trees to some transportation system.

As a matter of fact, the repair policy models were effective vehicle of demonstrating the language's expressive power, which enabled us to model even more repair policies than the ones outlined in this chapter. As such, we want to leverage FGTD usage among domain experts by combining them with Activity Diagrams, a practically appreciated aspect of UML, while investigating an airplane conservation problem.

References

[1] Ajmone Marsan, M., Balbo, G., Conte, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2, 93–122 (1984)

[2] Babczyński, T., Łukowicz, M., Magott, J.: Time coordination of distance protections using probabilistic fault trees with time dependencies. IEEE Transaction on Power Delivery 25(3), 1402–1409 (2010)

[3] Bobbio, A., Codetta, D.: Parametric fault trees with dynamic gates and repair boxes. In: Proc. Annual Symposium on Reliability and Maintainability, pp. 459–465 (2004)

[4] Codetta, D., Franceschinis, G., Iacono, M., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: Proc. Proceedings of the International Conference on Dependable Systems and Networks, DSN 2004(2004)
[5] Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. 41(3), 363–367 (1992)
[6] International Technical Commission, Fault Tree Analysis (FTA), Publication 1025 (1990)
[7] Górski, J.: Extending Safety Analysis Techniques with Formal Semantics, Technology and Assessment of Safety-Critical Systems, pp. 147–163. T. Springer-Verlag, Heidelberg (1994)
[8] Górski, J., Magott, J., Wardzinski, A.: Modelling fault trees using Petri nets. In: Proc. SAFECOMP 1995, Belgirate, Italy. LNCS. Springer, Heidelberg (1995)
[9] ISO/IEC 15909-1, High-level Petri nets: Concepts, definitions and graphical notation (2004)
[10] Kowalski, M., Magott, J.: Fault graphs with time dependencies (in Polish). Problemy eksploatacji, Maintenance Problems (1), 117–128 (2011)
[11] Magott, J., Skrobanek, P.: A Method of Analysis of Fault Trees with Time Dependencies. In: Koornneef, F., van der Meulen, M.J.P. (eds.) SAFECOMP 2000. LNCS, vol. 1943, pp. 176–186. Springer, Heidelberg (2000)
[12] Magott, J., Skrobanek, P.: Method of time Petri net analysis for analysis of fault trees with time dependencies. In: IEE Proceedings - Computers and Digital Techniques, vol. 149(6), pp. 257–271 (2002)
[13] Montani, S., Portinale, L., Bobbio, A., Codetta-Raiteri, D.: RADYBAN: a tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. In: Reliability Engineering and System Safety, vol. 93(7), pp. 922–932 (2008)
[14] RELEX, Resources on Fault Trees (2010), http://www.relex.com/resources/art

Analysis of Geometric Features of Handwriting to Discover a Forgery

Henryk Maciejewski and Roman Ptak

Wrocław University of Technology,
ul. Wybrzeże Wyspiańskiego 27,
50-370 Wrocław, Poland
e-mail: {Henryk.Maciejewski,Roman.Ptak}@pwr.wroc.pl

Abstract. This work proposes a method of analysis of geometric features generated from hand-written text to verify a supposition that a given sample of text of unclear authorship (e.g., a signature or initials) and some given reference text of known authorship have been written by the same author. The method is targeted to problems where the reference material is relatively large and the sample of unclear authorship is small, hence the number of feature vectors for the two groups compared is highly unbalanced. This makes the problem computationally challenging as standard approaches based on statistical hypothesis testing to compare distributions cannot be used. We propose a method to estimate the likelihood that the set of features observed in the small sample comes from the distribution generated from the reference material. This approach can be used to help discover or prove a forgery in documents.

1 Introduction

Analysis of handwritten text has been an important application area of machine learning or advanced statistical pattern recognition since early years of these disciplines. The purpose of analysis has been primarily the *recognition* of handwriting or personal *identification* of the writer based on hand-writing [4,6]. Various approaches to text recognition and writer identification were proposed in literature, e.g., based on Hidden Markov Models (HMM), Support Vector Machines with specialized kernels, or based on combining several different classifiers, [1,2,7,8]. Two different approaches to recognition were developed: *offline* or *online*, depending on whether features are derived only from the handwritten text (offline methods), or maybe also based on the analysis of the very process of writing (online methods).

The purpose of the method developed in this work is the detection of *forged hand-writing*, e.g., forged signatures or initials, etc. We assume that a signature (or a few signatures) is available, for which the authorship is unclear or questioned. We denote this hand-written text as the Questioned Material (QM). We also assume that a large reference sample is available for which the authorship is known; we denote this as the Reference Material (RM). QM and RM are schematically

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 145–154.
springerlink.com

illustrated in Fig. 1 and 2. The proposed method aims to verify the hypothesis that the QM and RM were written by the same author. We develop an offline method based on the analysis of geometric features of handwriting. It should be noted that in this problem formulation, the number of features derived from the QM is inevitably small, while the number of features from the RM is large enough to allow for estimation of the distribution of features in RM. The method proposed is intended to compare features between such highly unbalanced classes.

In the following section, we define the set of geometric features that can be computed on the basis of short hand-written texts, such as signatures or initials. Next we propose a method to verify the hypothesis that feature vectors generated from QM and RM come from the same distribution. If the hypothesis proves true, we conclude that QM was not forged, i.e. it was written by the person who wrote RM, otherwise QM is interpreted as forged. We provide a numerical example to illustrated this approach.

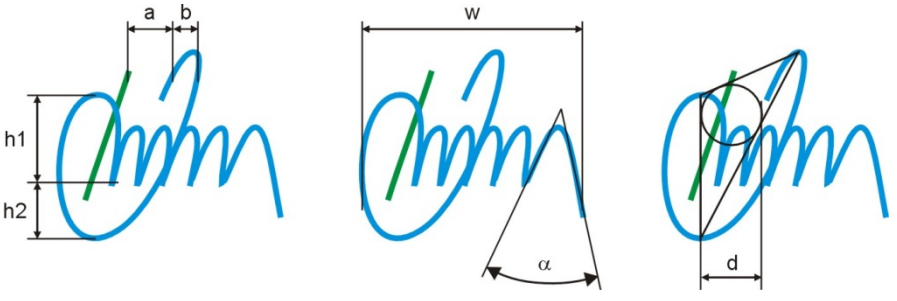


Fig. 1 The initials model of the questioned material (QM) and measured features (explanation in text)

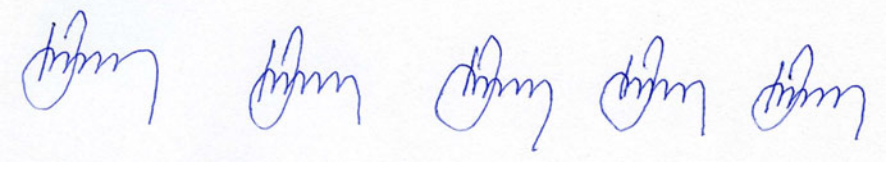


Fig. 2 A sample of text in the reference material (RM)

2 Geometric Features Derived from Handwritten Text

When performing a handwriting examination and comparison, many features of the handwriting are taken into consideration. It is possible to obtain features at the following document levels: basic, macrostructural and microstructural. The Catalogue of Graphic Features of Handwriting distinguishes five groups of features: synthetic, topographic, motor, measurable and constructional [3,5]. Some of the features of handwriting are: the overall size of the writing, the width of letters, words, etc., pen lifts within and between letters, the curvature of pen strokes [4]. Geometric features are parts of topographic features and measurable features on

basic document level. Some of these features are used also for examination of long texts but are also applicable for short texts (such as initials), examples are shown in Figs. 3 and 4. Other features are specific to examination of initials (e.g., the radius of rotation of the hand, illustrated in Fig. 1 and explained in the following paragraph).

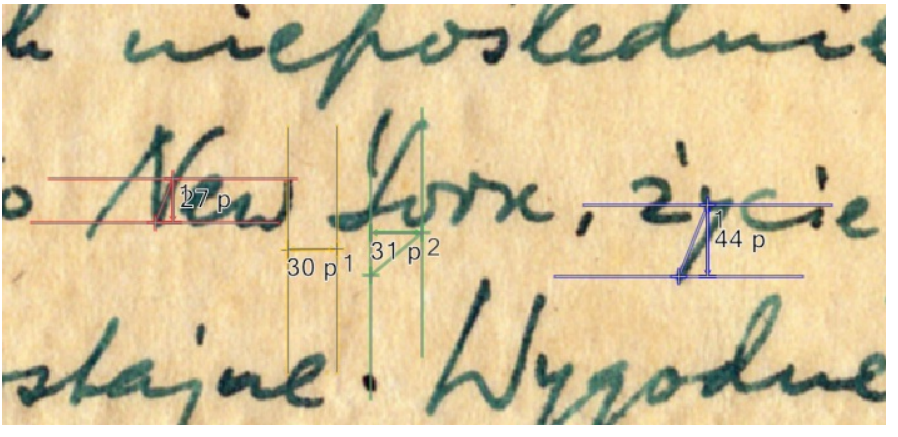


Fig. 3 Examples of features measured from long text: the size of handwriting (the high of middle zone), the high of lower zone, distance of words and distance between strokes in pixels

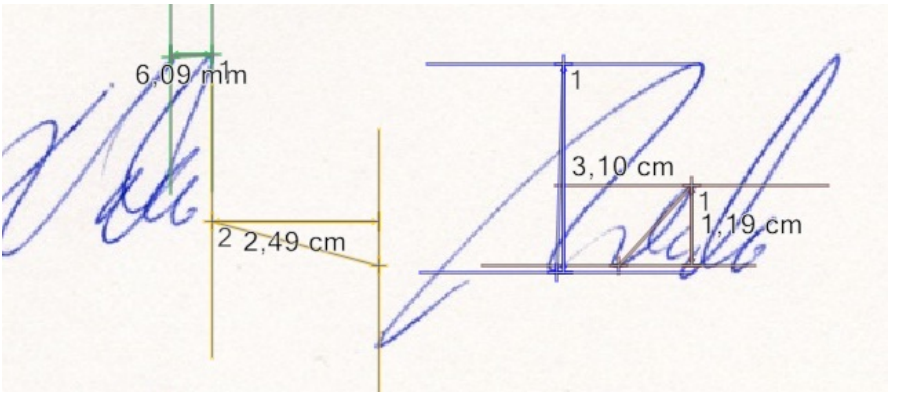


Fig. 4 Example of features measured from short text (two initials): the high of middle zone, the high of upper zone, distance of words and distance between strokes in mm and cm

The proposed method of analysis of short text will be illustrated using the sample initials shown in Fig. 1. The initials consist of two strokes. The variable a represents between strokes distance and b represents inner main strokes distance. Feature h1 is the high of middle zone and h2 is the high of lower zone. The next three extreme points of the main stroke of initials mark the limits of pen trajectory. The inscribed circle in the triangle represents the radius of rotation of the hand.

This feature is also dependent on the human anatomy. We also measure the angle α between final strokes of initials. Proportions between the various features of handwriting (ratios) are generally considered most informative.

We propose to use the following geometric features to compare the questioned material with the reference material:

- Feature 1: f_1=a/b is the ratio of the outer (a) to inner (b) strokes distance,
- Feature 2: f_2=h1/h2 is the ratio of the high of middle zone (h1) to the high of the lower zone (h2),
- Feature 3: f_3=α is the angle of the final strokes,
- Feature 4: f_4=w/d is the ratio of the initials width (w) to the diameter (d) of an inscribed circle of a triangle determined by the extreme points in main stroke of initials.

In order to illustrate the proposed method we will analyze the set of 120 initials in the RM class and two initials in the QM class as schematically shown in Figs. 1 and 2. The geometric features were measured for these texts in a semiautomatic way using the developed software application shown in Fig. 5.

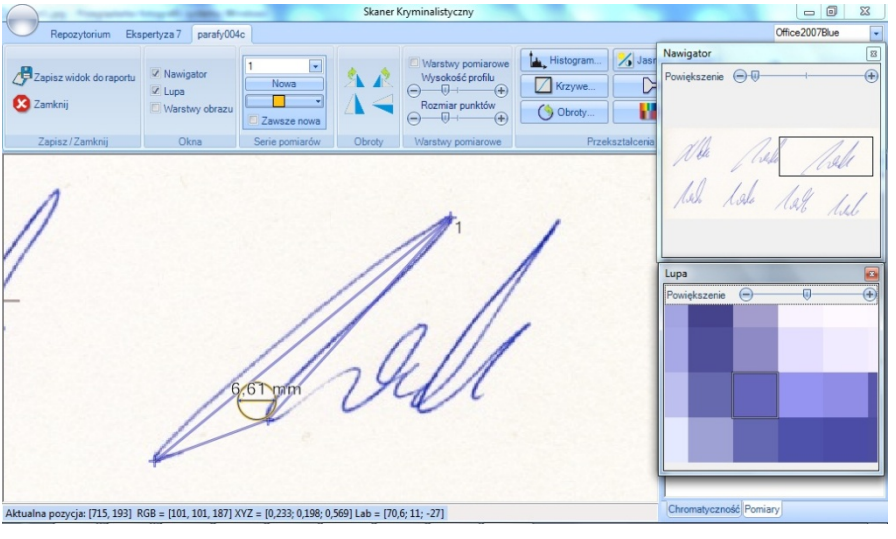


Fig. 5 Measurement window of the software application developed to facilitate examination of documents

Values of the proposed geometric features calculated from the two initials in the QM group are summarized in Table 1.

Table 1 Geometric features calculated from the two signatures in QM

Initials	A	b	**a/b**	h1	h2	**h1/h2**	**α**	w	D	**w/d**
QM1	180	260	**0.69**	503	517	**0.972**	**52.1**	1470	574	**2.56**
QM2	198	266	**0.74**	594	278	**2.136**	**49.1**	1386	584	**2.37**

In Fig. 6 we provide a preliminary comparison of the individual features between the QM and RM groups. The distribution of a feature in RM is represented by a boxplot, with the box representing the inter-quartile range (IQR) and the whiskers spanning the range ±1.5 IRQ around the median. It can be observed that QM and RM are most differential in terms of the f_3 feature which is significantly larger in QM than in RM (more specifically, the value of f_3 for the two samples in QM lie in the outlier area of the distribution of RM).

In the next section we propose a measure which aggregates the similarities between QM and RM in terms of individual features into a value of likelihood of randomly selecting a given QM sample from the distribution of RM samples.

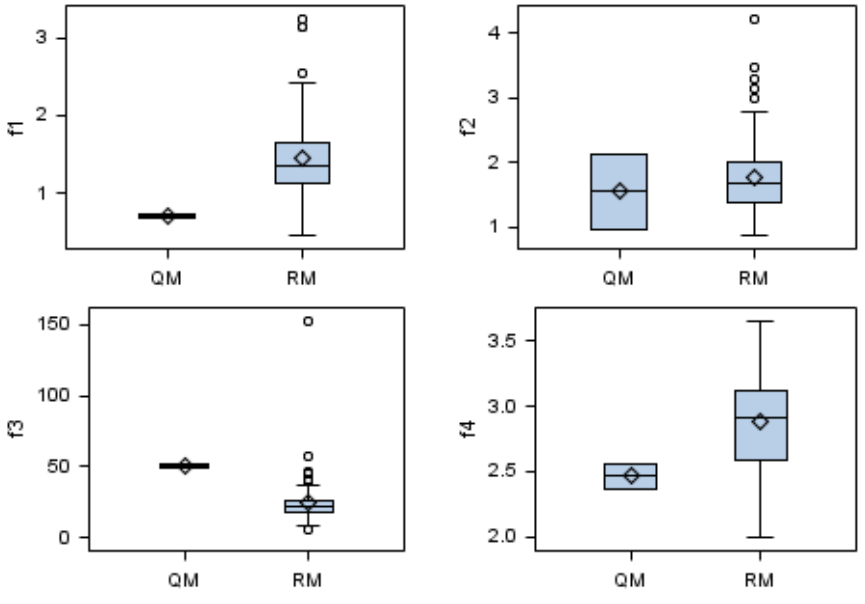


Fig. 6 Values of features from two samples of the questioned material (QM) shown in Table 1 compared with distributions of features calculated from the reference material (RM)

3 Verification of the Hypothesis of Common Authorship of QM and RM

In this section we propose a method to estimate the probability that a given sample (initials) from the QM group can be observed in the RM group. This measure is based on the features f_1 to f_4 of the QM sample compared with the corresponding distributions in the RM class.

We propose the following procedure.

1. For each of the features f_i, i=1,…,4 of the given sample from QM, we estimate the probability (denoted pc_i) that *the value f_i or a more extreme value* can be observed in the distribution for the RM class:

$$pc_i = \begin{cases} q_i & dla\ q_i < 0.5 \\ 1 - q_i & dla\ q_i \geq 0.5 \end{cases}$$

where $q_i = F_i(f_i)$, and F_i denotes the cumulative distribution function of the feature i estimated for RM. The value of pc calculated for q<0.5 is illustrated in Fig. 7 (pc_1=0.017, calculated for initials QM1, this value is represented by left tail shaded in red). The value of pc corresponding to q>0.5 is illustrated by the right tail in Fig. 8 (pc_3=0.12 calculated for initials QM1).

2. We estimate the joint probability that the set of features f_1-f_4 for the QM sample can be observed in the RM class:

$$p = \prod_{i=1}^{4} pc_i$$

This formula resembles the rule used in naïve Bayes classifier, where the features are assumed as independent. In this example, we observe that the features f_1-f_4 are weakly correlated (with the correlation coefficient ranging from 0.026 to 0.16) and realize very similar eigenvalues of the feature correlation matrix (results not shown) – this indicates weak association between features.

3. We calculate the number of samples (initials) in the RM class for which the value p (computed according to the formula in step 2) does not exceed the value p computed for the given QM sample. This number divided by the total number of samples in RM can be interpreted as the likelihood that the sample QM can be observed in the distribution of RM, under the hypothesis that QM comes from the distribution RM. We denote this as pVal:

$$pVal = \frac{1}{|RM|} \sum_{r \in RM} I(p_r \leq p)$$

where p_r is calculated for a sample r∈RM according to step 2, and p is calculated for the sample QM according to step 2. The function I returns 1 if the condition is true, and 0 – otherwise.

To illustrate the way to verify the hypothesis that the QM initials have been written by the author of RM, we summarize the values q_i, p and pVal calculated by the proposed procedure in Table 2. With the confidence level of 5% we conclude that the sample QM2 was *not* written by the author of RM (as pVal<0.05 for QM2,

Table 2 The value of q for the initials in the QM class, and the probability of QM initials occurring in the RM class

Initials	q_1	q_2	q_3	q_4	p	**pVal**
QM1	0.017	0.287	0.88	0.18	0.00011	**0.0952**
QM2	0.029	0.822	0.87	0.07	4.82E-5	**0.0397**

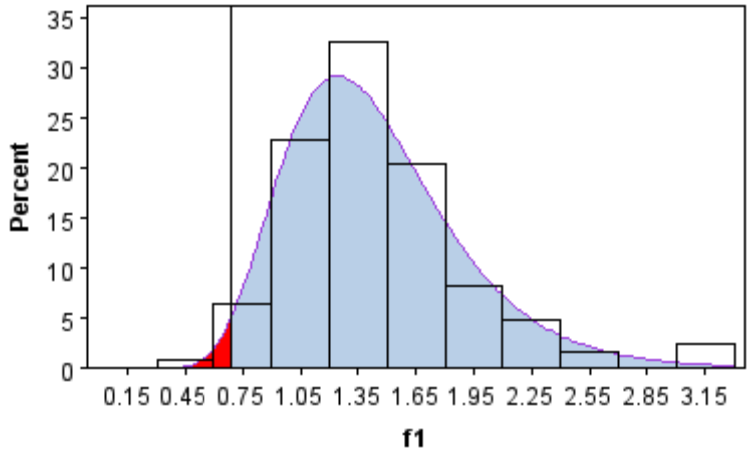


Fig. 7 Illustration of the value pc_1 for feature f_1 calculated for the initials QM1 from the distribution of this feature for RM. The pc value corresponds to the left tail of the distribution (marked red)

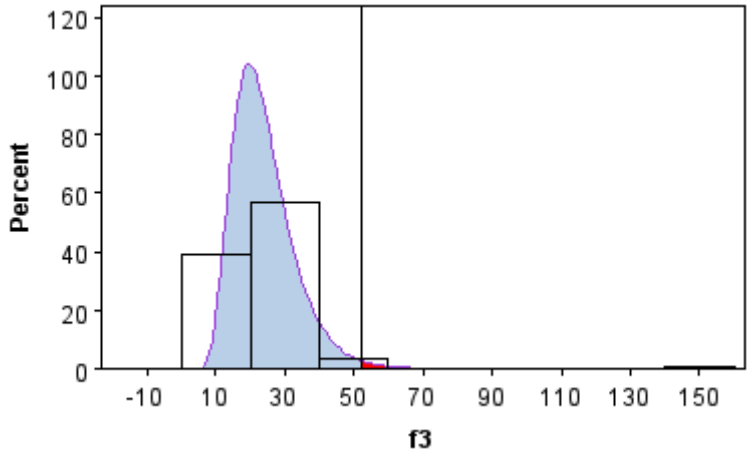


Fig. 8 Illustration of the value pc_3 for feature f_3 calculated for the initials QM1 from the distribution of this feature for RM (the pc values)

hence the hypothesis of common authorship is rejected). However, this conclusion cannot be formulated for the sample QM1 (as pVal > 0.05).

The pVal indicates how unlikely the value of p calculated for the QM sample is in the distribution pertaining to the RM class. This can be also shown graphically – see Fig. 9, where the p for QM (given in log scale) is compared with the distribution for RM.

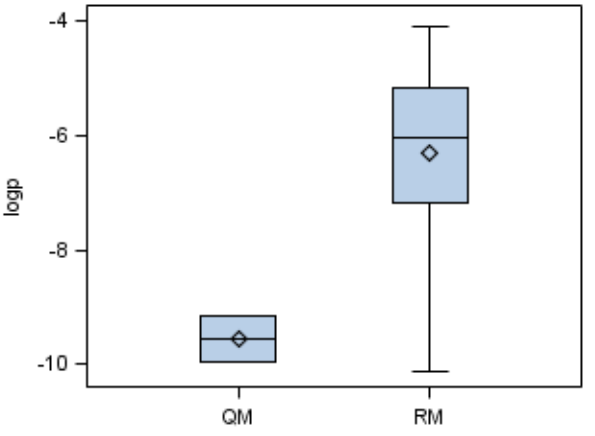


Fig. 9 The measure p calculated for the QM samples compared with the distribution for the RM group

The analyses discussed in this section and summarized in Table 2, Figs. 9 and 6 provide numerical and graphical indications about how *different* the questioned material seems to be as compared with the reference material. However, this approach is unable to discover text forgeries which consist in taking an (almost exact) copy of reference initials (using e.g., photocopying techniques). The method developed in the next section aims to discover this type of forgery by revealing that the questioned material should be *too similar* to the reference material.

4 Analysis of Similarity of Feature Vectors

In this section we extend the previous analysis by directly comparing feature vectors between the classes QM and RM. The purpose of this is to measure the *natural variability* of features in the RM group, characteristic of the set of initials written by a human (and not a machine). Then, if some of the QM initials are *too similar* to some of the RM samples, this clearly indicates possibly forged handwriting.

This idea motivates the following procedure.

1. For each of the samples $r_i \in RM$ from the reference group RM, calculate the distance $dmin_i$ to the nearest sample from RM:

$$dmin_i = \min_{j \neq i} \{dist(r_i, r_j) : r_j \in RM\}$$

   In the following example we use the Euclidean distance $dist(r_i, r_j)$ between the feature vectors.
2. For the QM sample $qm \in QM$ calculate the minimum distance $dmin_{qm}$ to the samples from RM, i.e.

$$dmin_{qm} = \min_{j} \{dist(qm_i, r_j) : r_j \in RM\}$$

3. Estimate the probability pr= G($dmin_{qm}$) that the value $dmin_{qm}$ *or smaller* is observed in the distribution of dmin calculated from RM samples. If this probability is small, the QM sample seems *too similar* to some of the RM samples then expected taking into account the natural variability among RM samples. Hence the QM sample is presumably a copy of a RM sample (hence forged?).

   If pr is *not* small, then calculate the probability pr2=1- G($dmin_{qm}$) that the value $dmin_{qm}$ *or bigger* is observed in the distribution of dmin calculated from RM samples. If this probability is small, the QM sample seems *too different* from the RM samples then expected under the hypothesis that QM and RM samples come from the same distribution. Hence the QM sample is presumably written by a different author than RM.

To illustrate this, in Fig. 10, the distribution of dmin for the RM class is shown as a boxplot (right part of Fig. 10). We conclude that if a QM sample was a copy of a RM sample (i.e. dmin≈0), this would be immediately clear as the value of dmin≈0 is below the lower tail of the distribution.

The left part of Fig. 10 shows the values of dmin calculated for the two samples QM1 and QM2 analyzed in section 3. We clearly see that the value pr2 (see step 3) for these samples equals about 0 (as the values are far above the upper tail of the distribution). Thus we conclude that these two questioned samples were presumably written by a different author than RM.

This result is generally consistent with results obtained in section 3 (although here the QM1 sample is classified as different than RM, while the test in section 3 would need significance level of 0.1 to confirm this).

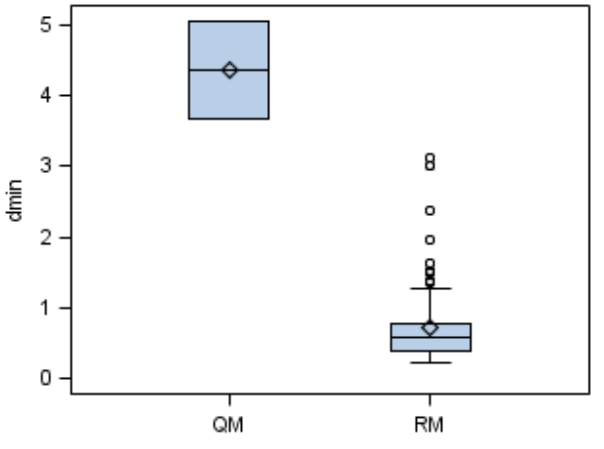


Fig. 10 Distribution of the distance to the nearest neighbor sample in the RM group (dmin) shown for the RM group (right boxplot), and distance of QM samples to the nearest RM sample (left boxplot)

5 Conclusions

The methods proposed in this work can be used to provide quantitative and graphical indications whether questioned text (such as initials) was written by the author of reference material. The analysis is based on the set of four geometric features calculated from samples of handwriting. The indications are based on probabilistic analysis of variability of features calculated from the reference material. The questioned initials are considered significantly different than the reference initials (which may indicate different authorship) if their features occupy the far right tail of the distribution characteristic of the reference group. We also propose a method to discover initials which are *too similar* to the reference material than expected taking into consideration natural variability in hand-written text. Such too similar initials may be deemed suspicious or forged.

However, it should be made clear that all results shown in this work are of probabilistic nature. As such, they should not be directly translated into a firm statement about some questioned material being forged. Such conclusions are to be made only by a trained human evaluator (a forensic expert), for whom the methods elaborated here may provide some decision support data.

References

[1] Bahlmann, C., Haasdonk, B., Burkhardt, H.: Online handwriting recognition with support vector machines - a kernel approach. In: Proc of the Eighth International Workshop on Frontiers in Handwriting Recognition, pp. 49–54 (2002), doi:10.1109/IWFHR.2002.1030883

[2] Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning – Data Mining, Inference, and Prediction. Springer, Heidelberg (2001)

[3] Katalog Graficznych Cech Pisma Ręcznego (The Catalogue of Graphic Features of Handwriting) (in Polish) (2007), `http://prawo.amu.edu.pl/uploads/slownik/aneks.htm` (accessed March 12, 2011)

[4] Koziczak, A.: Metody pomiarowe w badaniach pismoznawczych (Measurement methods in examination of handwriting, in Polish), Instytut Ekspertyz Sądowych, Kraków (1997)

[5] Morris, R.: Forensic handwriting identification. Fundamental concepts and principles. Academic Press, New York (2000)

[6] Saferstein, R.: Criminalistic: An Introduction to Forensic Science, 8th edn. Prentice-Hall, Englewood Cliffs (2004)

[7] Schlapbach, A., Bunke, H.: A writer identification and verification system using HMM based recognizers. Pattern Analysis & Applications 10(1), 33–43 (2007), doi:10.1007/s10044-006-0047-5

[8] Xu, L., Krzyzak, A., Suen, C.Y.: Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Trans on Systems, Man and Cybernetics 22(3), 418–435 (2002)

A Formal Framework for Testing duration Systems

Lotfi Majdoub

LIP2 Laboratory, Faculté des sciences de Tunis
e-mail: lotfi.majdoub@ensi.rnu.tn

Abstract. The aim of conformance testing is to check whether an implementation conforms to its specification. We are interested to duration systems; we consider a specification of a duration system that is described by a duration graph. Duration graphs are an extension of timed automata and are suitable for modeling the accumulated times spent by computations in duration systems. In this paper, we propose a framework to generate automatically test cases directly from the specification model. In the first, in order to reduce the infinite set of states due to the continuous time, we use a digitization technique. Test cases are given by considering a discrete time and represented with a tree. In the second, we demonstrate that every test cases derived from the test tree corresponds to a digitization test case of the specification model.

1 Introduction

Testing is an important validation activity. In most cases, it is used to check whether an implementation, referred to as an Implementation Under Test (IUT), conforms to its specification.

Duration systems [6][3] are an extension of real-time systems for which in addition to constraints on delays separating certain events that must be satisfied, constraints on accumulated times spent by computation must also be satisfied. Duration systems are used in a large number of applications such as real-time scheduler systems, communication protocols, distributed systems etc.

Duration systems are characterized by a number of generic features:

- Due to the interaction with their environment, duration systems must fulfill strict temporal requirements. The correct behavior of these systems depends not only on the logical results of the computation but also on the time at which these results are produced. So, for those systems, a continuous time is considered.
- In systems where the time progresses continuously, the space of states is infinite. Many different techniques have been proposed to reduce the infinite set of states to a finite set (or at least countable) (e.g., symbolic

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 155–168.
springerlink.com

techniques, region graph and its variations, model checking techniques, etc.) [7][10][9][11]. Due to the accumulation of times in duration systems, it has been shown [5] that it is not possible to use those techniques both in verification and in test of duration systems.

Model-based testing [4] is a testing method that consists to describe the behaviour of a system with a model and generating test cases from this model. The main advantage of model-based testing is the facility to derive automatically test cases.

Timed automata constitute a powerful formalism widely adopted for modelling real-time systems [2]. Duration Variables Timed Graphs with Inputs Outputs (DVTG-IOs) are an extension of timed automata [3], which are used as a formalism to describe duration systems. DVTG-IOs are supplied with a finite set of continuous real variables that can be stopped in some locations and resumed in other locations [6]. These variables are called *duration variables*. They can model some temporal behaviours of real-time systems such as the accumulated times spent by computations at some particular locations.

The contributions of this chapter are twofold. First, we give a framework for generating automatically test cases directly from the specification model. In order to reduce the infinite set of states due to the continuous time, we use a digitization technique. Test cases are given by considering a discrete time and represented with a tree. Second we demonstrate that every test case derived from the test tree corresponds to a digitization test case of the specification model.

This chapter is organised as follows: In the next section, we present the duration variables timed graphs with inputs outputs used to model specification. Section 3 shows the digitization technique. The test generation framework is introduced in section 4. Section 5 presents properties of test cases that are derived with our framework. Concluding remarks are presented n section 6.

2 Duration Variables Timed Graphs with Inputs Outputs

In this section, we introduce the formalism that we use for describing duration systems, called Duration Variables Timed Graph with Inputs Outputs (DVTG-IO for short). DVTG-IO is an extension of the well known timed automata defined in [2].

We give here the formal definition and the operational semantics of this model. Then we illustrate with an example.

2.1 Formal Definition

A DVTG-IO is described by a finite set of locations and a transition relation between these locations. In addition, the system has a finite set of duration variables that are constant slope continuous variables, each of them changing continuously

with a rate in {0, 1} at each location of the system. Transitions between locations are conditioned by arithmetical constraints on the values of the duration variables. When a transition is taken, a subset of duration variables should be reset and an action should be executed. This action can be an input action, an output action or an unobservable action (known also as quiescent [17]).

We consider X a finite set of duration variables. A guard on X is a Boolean combination of constraints of the form $x \prec c$ where $x \in X, c \in \mathbb{N}, \prec \in \{<, \leq, >, \geq\}$. Let $\Gamma(X)$ be the set of guards on X,

A DVTG-IO describing duration systems is a tuple $S = (Q, q_0, E, X, Act, \gamma, \alpha, \delta, \partial)$ where Q is a finite set of locations, q_0 is the initial location, $E \subseteq Q \times Q$ is a finite set of transitions between locations, $Act = Act_{In} \cup Act_{Out}$ is a finite set of input actions (denoted by *?a*) and output actions (denoted by *!a*), $\gamma: E \to \Gamma(X)$ associates to each transition a guard which should be satisfied by the duration variables whenever the transition is taken, $\alpha: E \to 2^X$ gives for each transition the set of duration variables that should be reset when the transition is taken, $\delta: E \to Act$ gives for each transition the action that should be executed when the transition is taken, $\partial: Q \times X \to \{0,1\}$ associates with each location q and each duration variable x the rate at which x changes continuously while the computation is at q.

2.2 Example

We give, in figure 1, a simple example that illustrates Duration Variables Timed Graph with Inputs Outputs. It is composed of locations $\{q_0, q_1, q_2, q_3, q_4\}$ where q_0 is the initial location and transitions between locations. In this figure, DVTG-IO is supplied with a set of input actions *{?a, ?c}*, output actions *{!b, !d, !r}*. Duration variables x and y are clocks used to make constraints on the time execution, z is a duration variable, it is stopped ($\dot{z} = 0$) in locations q_1, q_3.

2.3 State Graph

The semantic of DVTG-IO is defined in terms of a state graph over states of the form $s = (q, v)$ where $q \in Q$ and $v: X \to \mathbb{R}^+$ is a valuation function that assigns a real value to each duration variable. Let St_S be the set of states of S. We notice that St_S is an infinite set due to the value of duration variables taken on $\mathbb{R}^+$. A state (q, v) is called integer state if $v: X \to \mathbb{N}$. We denote by $N(St_S)$ the set of integer states of S.

Given a valuation v and a guard g, we denote by $v \vDash g$ the fact that the evaluation of g under the valuation v is true.

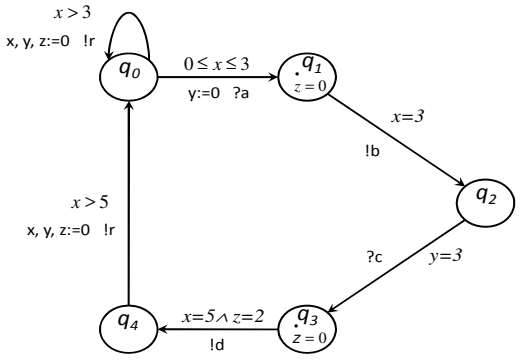


Fig. 1 Sample DVTG-IO

We define two families of relation between states:

– Discrete transition $(q,v) \rightsquigarrow^{a} (q',v')$ where $(q,q') \in E$, $\delta(q,q') = a$, $v \vDash \gamma(q,q')$ is true and $v'(x) = v(x)\ \forall x \in X \backslash \alpha(q,q')$, $v'(x) = 0\ \forall x \in \alpha(q,q')$ corresponds to moves between locations using transition in *E*.

– Timed transition $(q,v) \rightsquigarrow^{t} (q',v')$ such that $t \in \mathbb{R}^{+}$ and $v'(x) = v(x) + \partial(q,x) * t\ \forall x \in X$, corresponds to transitions due to time progress at some location q.

The state graph associated with *S* is (St_S, $\rightsquigarrow$) where $\rightsquigarrow$ denotes the union of all discrete transitions and timed transitions.

2.4 Basic Definitions

2.4.1 Computation Sequence

A computation sequence of a DVTG-IO is defined as a finite sequence of configurations. A configuration is a pair (s,τ) where *s* is a state and τ is a time value. Let C_S be the set of configurations of *S*. A configuration (s,τ) is called integer configuration if $\tau \in \mathbb{N}$. We denote by $N(C_S)$ the set of integer configurations.

Intuitively, a computation sequence is a finite path in the state graph of an extension of *S* by an observation clock that records the global elapsed time since the beginning of the computation. Formally, if we extend each transition relation from states to configurations, then a computation sequence of *S* is $\sigma = (s_0,0) \rightsquigarrow (s_1,\tau_1) \rightsquigarrow \cdots \rightsquigarrow (s_n,\tau_n)$, where $s_i = (q_i,v_i)$ and $\tau_{i-1} \leq \tau_i$ for $i = 1,\ldots,n$. Let *CS(S)* be the set of computation sequences of *S*.

2.4.2 Timed Word

A timed word is a finite sequence of timed actions. A timed action is a pair $a\tau$ where $a \in Act$ and $\tau \in \mathbb{R}^+$, meaning that the action *a* takes place when the observation clock is equal to τ. A timed action $a\tau$ is called integer timed action if $\tau \in \mathbb{N}$. A timed word is a sequence $\omega = a_1\tau_1a_2\tau_2 \dots a_n\tau_n$ where a_i is an action and τ_i is a value of the observation clock. We notice that $\tau_i \leq \tau_{i+1}$. Let *L(S)* be the set of timed words of *S*.

We can easily make the link between timed word and computation sequence. Clearly, there exists a unique timed word ω corresponding to a computation sequence σ. Let $\sigma = (s_0, 0) \rightsquigarrow (s_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (s_n, \tau_n) \in CS(S)$ be a computation sequence, its corresponding timed word $\omega = a_1\tau_1a_2\tau_2 \dots a_n\tau_n \in L(S)$ is obtained such that $a_i = \delta(q_{i-1}, q_i)$ and τ_i is the value of the observation clock in the state s_i for $i = 1, \dots, n$ and $s_i = (q_i, v_i)$. On the other hand, if $\omega = a_1\tau_1a_2\tau_2 \dots a_n\tau_n \in L(S)$ is a timed word then there exists at least one computation sequence $\sigma = (s_0, 0) \rightsquigarrow (s_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (s_n, \tau_n) \in CS(S)$ such that $a_i = \delta(q_{i-1}, q_i)$ for $i = 1, \dots, n$ and $s_i = (q_i, v_i)$.

Consequently, a sequence $\omega = a_1\tau_1a_2\tau_2 \dots a_n\tau_n$ is considered a timed word of $L(S)$ if and only if there exists a computation sequence $\sigma = (s_0, 0) \rightsquigarrow (s_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (s_n, \tau_n) \in CS(S)$ such that $a_i = \delta(q_{i-1}, q_i)$ for $i = 1, \dots, n$ and $s_i = (q_i, v_i)$. For simplicity, we may write $(s_0, \tau_0) \rightsquigarrow^{\omega} (s_n, \tau_n)$. We suppose that empty timed word belongs to $L(S)$.

Let $\omega = a_1\tau_1a_2\tau_2 \dots a_n\tau_n$ be a timed word and $a \in Act$ and $\tau \in \mathbb{R}^+$, such that $\tau_n \leq \tau$ then we denote by $\omega.a\tau$ the timed word obtained by adding $a\tau$ to ω and we have $\omega.a\tau = a_1\tau_1a_2\tau_2 \dots a_n\tau_n a\tau$.

2.4.3 The Restart Function

A restart of a variable *x* is the change of its rate from 0 to 1. For example, if in a location *q* the rate of a variable *x* is 0 and there exists a transition (q, q') such that the rate of *x* in *q'* is 1 then we consider, in location *q'*, a restart of *x*.

We define the restart function, as a function that calculates the number of restarts of one duration variable in a computation sequence.

Formally, the restart function is $\beta: X \times CS(S) \rightarrow \mathbb{N}$. $\beta(x, \sigma)$ calculates for each variable $x \in X$ and each computation sequence $\sigma = (s_0, 0) \rightsquigarrow (s_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (s_n, \tau_n) \in CS(S)$ the number of restarts of *x* from the last reset of *x* until the location q_n in σ.

After a reset of a variable *x*, if the rate of a variable *x* in the current location is 1, then the access to this location is considered as a restart of *x*. For example, if in the first location *x* has a rate equal to 1 then the access to the initial state is considered as restart. That is why, for the clocks, the function β is equal to 1 for each transition.

3 Digitization

We present the notion of digitizations [8], which is suitable for systems in which we are interested. It is used particularly to reduce the infinite set of states to a countable set of states.

Let us introduce some definitions and notations. Let $\tau \in \mathbb{R}^+$, for every $\varepsilon \in [0,1[$ called *digitization quantum*, we define the integer:

$$[\tau]_\varepsilon = if\ \tau \leq (\lfloor \tau \rfloor + \varepsilon)\ then\ \lfloor \tau \rfloor\ else\ \lceil \tau \rceil$$

from this definition, we deduce that

$$\tau \in\ ]\,[\tau]_\varepsilon - 1 + \varepsilon, [\tau]_\varepsilon + \varepsilon[$$

and

$$-1 + \varepsilon < \tau - [\tau]_\varepsilon \leq \varepsilon$$

Before giving the definition of the digitization of a computation sequence, let us introduce how we can calculate at each step the valuation of variables. Given a computation sequence σ.

$$\sigma = (q_0, \nu_0, \tau_0) \rightsquigarrow (q_1, \nu_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (q_n, \nu_n, \tau_n)$$

Let x be a duration variable of X. The value of x at position k in σ is:

$$\nu_k(x) = \sum_{i=j_k+1}^{k-1} \partial(q_i, x) * (\tau_{i+1} - \tau_i)$$

Where j_k denotes the greatest index j such that $j \leq k$, and the transition $\rightsquigarrow_j$ is of the form $\rightsquigarrow^a$ with $x \in \alpha(q_j, q_{j+1})$ i.e., x is reset by (q_j, q_{j+1}). We take $j_k = -1$ if such an index does not exist.

According to [8], given a digitization quantum $\varepsilon \in [0,1[$, the digitization of σ is the integer computation sequence

$$[\sigma]_\varepsilon = (q_0, \nu_0^\varepsilon, [\tau_0]_\varepsilon) \rightsquigarrow (q_1, \nu_1^\varepsilon, [\tau_1]_\varepsilon) \rightsquigarrow \cdots \rightsquigarrow (q_n, \nu_n^\varepsilon, [\tau_n]_\varepsilon)$$

Where the valuation of a variable x at position k in $[\sigma]_\varepsilon$ is:

$$\nu_k^\varepsilon(x) = \sum_{i=j_k+1}^{k-1} \partial(q_i, x) * ([\tau_{i+1}]_\varepsilon - [\tau_i]_\varepsilon)$$

The restart function is presented in [16]. Basically, it calculates for each variable $x \in X$ and each transition the maximum of restarts of x from the last reset of x in each computation sequence of S. In the present chapter, we apply the restart function in one computation sequence instead of all computation sequences. We adapt the following result, first introduced in [16], and that makes the link between the restart function and the valuation function.

Proposition 1

For every computation sequence $\sigma = (q_0, \nu_0, \tau_0) \rightsquigarrow (q_1, \nu_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (q_n, \nu_n, \tau_n)$ of S, every quantum $\varepsilon \in [0,1[$, and every variable $x \in X$ we have:

- $\beta(x, \sigma) = 0 \Longrightarrow \nu_n^\varepsilon(x) - \nu_n(x) = 0$
- $\beta(x, \sigma) > 0 \Longrightarrow -\beta(x, \sigma) < \nu_n^\varepsilon(x) - \nu_n(x) < \beta(x, \sigma)$

Proof

For every computation sequence σ of S and every $x \in X$, we have

$$v_n^{\varepsilon}(x) - v_n(x) = \sum_{i=j_n+1}^{n-1} \partial(q_i, x) * ([\tau_{i+1}]_{\varepsilon} - [\tau_i]_{\varepsilon}) - \sum_{i=j_n+1}^{n-1} \partial(q_i, x) * (\tau_{i+1} - \tau_i)$$

$$v_n^{\varepsilon}(x) - v_n(x) = \sum_{i=j_n+1}^{n-1} \partial(q_i, x) * (([\tau_{i+1}]_{\varepsilon} - [\tau_i]_{\varepsilon}) - (\tau_{i+1} - \tau_i))$$

Where j_n denotes the greatest index j such that $j \leq n$, and the transition $\rightsquigarrow_j$ is of the form $\rightsquigarrow^a$ with $x \in \alpha(q_j, q_{j+1})$ i.e., x is reset by (q_j, q_{j+1}). We take $j_n = -1$ if such an index does not exist.

For a given variable x, if $\beta(x, \sigma) = 0$ then the rate of x is 0 in locations visited from the last reset of x ($\partial(q_i, x) = 0 \; for \; i = j_n, \dots, n$). So we have $v_n^{\varepsilon}(x) - v_n(x) = 0$.

For a given index n, a given transition (q_{n-1}, q_n), and a given variable x, if $\beta(x, \sigma) > 0$ then the computation sequence σ has a unique decomposition into a finite number of sets of indices $\{0, \dots, j_0 - 1\}$, $\{j_0, \dots, j_1\}$, $\{j_1 + 1, \dots, j_2 - 1\}$, $\{j_2, \dots, j_3\}, \dots$, $\{j_p, \dots, n\}$ where:

– j_0 is the first index i where $\partial(q_i, x) = 1$ after the last reset of x

– $\partial(q_i, x) = 1$ in the following sets: $\{j_0, \dots, j_1\}, \{j_2, \dots, j_3\}, \dots, \{j_{2k}, \dots, j_{2k+1}\}$ and $\partial(q_i, x) = 0$ in the others sets.

We have $\beta(x, \sigma) = k + 1$ by definition of β, because the number of restarts of x is equal to the number of subsets of indices on which $\partial(q_i, x) = 1$. Then

$$v_n^{\varepsilon}(x) - v_n(x) = \sum_{i=j_0}^{j_1-1}(([\tau_{i+1}]_{\varepsilon} - [\tau_i]_{\varepsilon}) - (\tau_{i+1} - \tau_i)) + \dots + \sum_{i=j_{2k}}^{j_{2k+1}-1} (([\tau_{i+1}]_{\varepsilon} - [\tau_i]_{\varepsilon}) - (\tau_{i+1} - \tau_i))$$

Where j_0 is the first index i where $\partial(q_i, x) = 1$ after the last reset of x.

For each $m \in \{0,1, \dots, k\}$, we have:

$$-1 < \sum_{i=j_{2m}}^{j_{2m+1}-1} (([\tau_{i+1}]_{\varepsilon} - [\tau_i]_{\varepsilon}) - (\tau_{i+1} - \tau_i)) < 1$$

So we have:

$$-\beta(x, \sigma) < v_n^{\varepsilon}(x) - v_n(x) < \beta(x, \sigma)$$

□

Let us extend the definition of the digitization to the timed word. The digitization of a timed word

$$\omega = a_1\tau_1 a_2\tau_2 \dots a_n\tau_n$$

is

$$[\omega]_\varepsilon = a_1[\tau_1]_\varepsilon a_2[\tau_2]_\varepsilon \dots a_n[\tau_n]_\varepsilon$$

Therefore, it is not difficult to see that:

$$[\omega.at]_\varepsilon = [\omega]_\varepsilon.a[t]_\varepsilon$$

Moreover, it is easier to relate digitizations of a computation sequence and its timed word. If σ is a computation sequence and ω is its corresponding timed word then for $\varepsilon \in [0,1|$, $[\omega]_\varepsilon$ is the corresponding timed word of $[\sigma]_\varepsilon$.

We denote by $Digit(L(S))$ the set of all digitizations of all real timed words of S. We notice that $Digit(L(S))$ is countable.

The digitization is a technique used to reduce the infinite set of states to a finite set of states. A question that one may ask is whether $Digit(L(S)) \subseteq L(S)$ or not.

Example

If the following example, we will see that we can have a DVTG-IO with only one duration variable for which there exists a real timed word such that all their digitizations are not computation of the system.

Let ω be a real timed word of the DVTG-IO given in the figure 1

$\omega = ?a\ 1.5\ !b\ 3\ ?c\ 4.5\ !d\ 5$

There are two digitizations of ω:

$[\omega]_\varepsilon = ?a\ 1\ !b\ 3\ ?c\ 4\ !d\ 5$ with $0.5 < \varepsilon < 1$

$[\omega]_\varepsilon = ?a\ 2\ !b\ 3\ ?c\ 5\ !d\ 5$ with $0 \leq \varepsilon \leq 0.5$

It is easier to verify on the DVTG-IO given in figure 1 that ω is a timed word, however their two digitzations $[\omega]_\varepsilon$ for $0.5 < \varepsilon < 1$ and $0 \leq \varepsilon \leq 0.5$ are not timed words of the considered DVTG-IO.

4 Test Generation Framework

Here, we introduce our testing method for duration systems. We suggest reducing the infinite state graph associated to the specification model to a countable state graph.

First, let us introduce the notion of an observation which is a sequence of controllable (inputs) and observable (outputs) actions that are either executed or produced by the IUT followed with its occurrence time. Formally, we describe an observation by a timed word $\omega = a_1\tau_1 a_2\tau_2 \dots a_n\tau_n$ where $a_i \in Act$ and $\tau_i \in \mathbb{R}^+$ for $i = 1, \dots, n$.

Our result is based on a reduction of the infinite state graph of the specification model S to the countable state graph $(N(St_S), \rightsquigarrow^1 \cup \rightsquigarrow^a)$, where the space of states is reduced to the set of integer states. Transitions between states are either discrete transitions $(q, v) \rightsquigarrow^a (q', v')$ labeled with action in Act, or timed transitions $(q, v) \rightsquigarrow^t (q', v')$ labeled with a constant delay of time equal to 1. Notice that v and $v' \in [X \to \mathbb{N}]$.

4.1 The Test Tree

We use the countable state graph $(N(St_S), \rightsquigarrow^1 \cup \rightsquigarrow^a)$ to generate a finite set of test cases. This set of test cases is represented by a tree called *test tree*.

The test tree is composed by nodes that are integer configurations and transitions between those nodes. A node in the test tree is an integer configuration (s, τ) such that $s \in N(St_S), \tau \in \mathbb{N}$ and represents the possible current integer configuration of the IUT. The root is the initial configuration of $(N(St_S), \rightsquigarrow^1 \cup \rightsquigarrow^a)$ that is (s_0, τ_0). The transition between one node and its successor is labeled with a timed action $a\tau$ such that $a \in Act$ and $\tau \in \mathbb{N}$ and represents the timed action that can be executed when the transition is taken. A path from the root to one leaf of the tree represents a test case that is an integer timed word.

4.2 Algorithm of Generating Test Tree

We remember that the digitizations of all timed words *Digit(L(S))* are not included in $(N(St_S), \rightsquigarrow^1 \cup \rightsquigarrow^a)$. We give in this section an algorithm for constructing the test tree that contains digitizations of all timed words by considering both the countable state graph and the restart function.

The generating test tree algorithm, that we give below, can generate a test tree containing all digitizations timed word for every $\varepsilon \in [0,1[$ of all timed words of the specification model.

Before giving the generating test tree algorithm, we introduce some notations that we use later.

Definition 1: extended guard

For every transition $(q, q') \in E$, let $\gamma(q, q')$ its associated guard
Let $\beta(x, \sigma)$ the restart function such that $\sigma \in CS(S)$ and $x \in X$
The extended guard $\gamma(q, q')_{\beta_\sigma}$ is obtained from $\gamma(q, q')$ by transforming each constraint of the form $u \prec x \prec w$ by the constraint:
– if $u - \beta(x, \sigma) \geq -1$ then $u - \beta(x, \sigma) + 1 \leq x \leq w + \beta(x, \sigma) - 1$
– Otherwise $0 \leq x \leq w + \beta(x, \sigma) - 1$ where $u, w \in \mathbb{N}$, $x \in X$, and $\prec \in \{<, \leq\}$

□

Definition 2: test transition

Let $C = (q, v, \tau)$ and $C' = (q', v', \tau')$ be two integer configurations of $N(C_S)$
We define a test transition between C and C' that we denote with $(q, v, \tau) \rightsquigarrow^{a\tau'} (q', v', \tau')$ if there exists a configuration $C" = (q, v", \tau')$ such that:
– $(q, v, \tau) \rightsquigarrow^t (q, v", \tau') \rightsquigarrow^a (q', v', \tau')$,
– $(q, q') \in E$,
– $t \in \mathbb{N}$,
– $\tau' = \tau + t$,

– $a = \delta(q, q')$,
– $v" \vDash \gamma(q, q')_{\beta_\sigma}$,
– $v'(x) = v(x) + \partial(q, x) * (\tau' - \tau)\ \forall x \in X \backslash \alpha(q, q'), v'(x) = 0\ \forall x \in \alpha(q, q')$ □

```
1   Algorithm: Generating Test Tree
2   Input: the state graph (N(St_S), ⇝^1 ∪ ⇝^a)
3   Output: Test Tree T_S
4   T = {(s_0, 0)} the one-node tree
5   For each leaf C = (q, v, τ) of T_S distinct from pass do
6     Let σ = (q, 0,0) ⇝ (q, v, τ) the computation sequence from the root to C
7     Do randomly {1; 2}
8     Case 1:
9       For every (q, q') ∈ E, let γ(q, q')_{β_σ}
10        For every test transition C ⇝^{aτ} C'
11          Append edge C ⇝^{aτ} C' to T_S
12    Case 2:
13      C=pass
14  End
```

The generating test tree algorithm operates as follows: initially the test tree contains one node that is the initial configuration $(s, 0)$ of the countable state graph $(N(St_S), \rightsquigarrow^1 \cup \rightsquigarrow^a)$. For any leaf $C = (q, v, \tau)$ of the tree different from *pass*, and for each transition $(q, q') \in E$ the algorithm calculates its extended guard $\gamma(q, q')_{\beta_\sigma}$. Then, the test transition of this form $C \rightsquigarrow^{a\tau} C'$ is added to the tree. The algorithm can stop a path of the tree by appending the node *pass* in the leaf.

Example

Figure 2 illustrates the test tree of the specification model given in figure 1. This test tree is constructed according to the Generating Test Tree algorithm. Each path of the test tree from the root to one leaf is an integer computation sequence.

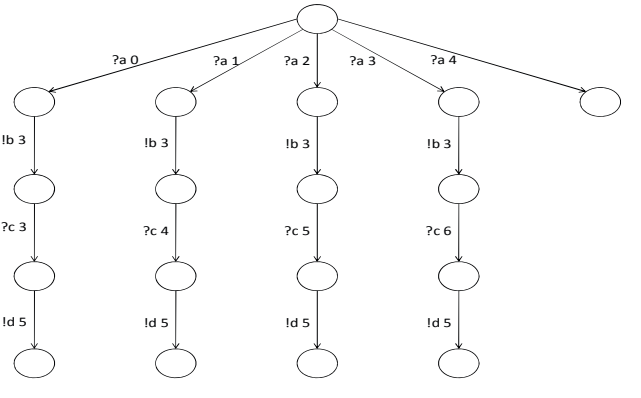


Fig. 2 The test tree

5 Properties of the Test Tree

In this section, we make the link between the digitization technique and the test tree. We demonstrate that a path from the root to one leaf in the test tree corresponds to a digitization of a timed word belonging to the specification model. In other hand, we demonstrate that given a timed word of the specification model, then their digitizations correspond to paths in the test tree.

Proposition 2

Let $\omega = a_1\tau_1 a_2\tau_2 \dots a_n\tau_n$ be a timed word and $\varepsilon \in [0,1[$
If $\omega \in L(S)$ then $[\omega]_\varepsilon \in T_S$

Proof

A sequence $\omega = a_1\tau_1 a_2\tau_2 \dots a_n\tau_n$ is considered a timed word of *L(S)* if and only if there exists a computation sequence $\sigma = (q_0, v_0, \tau_0) \rightsquigarrow (q_1, v_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (q_n, v_n, \tau_n) \in CS(S)$ such that $a_i = \delta(q_{i-1}, q_i)$ for $i = 1, \dots, n$.
Let $[\sigma]_\varepsilon$ be the digitization of σ for $\varepsilon \in [0,1[$,
$[\sigma]_\varepsilon = (q_0, v_0^\varepsilon, [\tau_0]_\varepsilon) \rightsquigarrow (q_1, v_1^\varepsilon, [\tau_1]_\varepsilon) \rightsquigarrow \cdots \rightsquigarrow (q_n, v_n^\varepsilon, [\tau_n]_\varepsilon)$
From proposition 1, we have $-\beta(x,\sigma) < v_i^\varepsilon(x) - v_i(x) < \beta(x,\sigma)$, with $i = 1, \dots, n$
We deduce that $-\beta(x,\sigma) + v_i(x) < v_i^\varepsilon(x) < \beta(x,\sigma) + v_i(x)$
We remember that $v_i(x)$ satisfy the guard of the transition (q_{i-1}, q_i), $v_i(x) \vDash \gamma(q_{i-1}, q_i)$
$\gamma(q_{i-1}, q_i)$ is a boolean combination of constraints of the form $u \prec x \prec w$
We have $-\beta(x,\sigma) + u < v_i^\varepsilon(x) < \beta(x,\sigma) + w$
Moreover, $v_i^\varepsilon(x)$ is an integer value. The previous formula can be written $-\beta(x,\sigma) + u + 1 \le v_i^\varepsilon(x) \le \beta(x,\sigma) + w - 1$
That corresponds to the extended guard, we have $v_i^\varepsilon(x) \vDash \gamma(q_{i-1}, q_i)_{\beta_\sigma}$
From the definition of the test tree, $[\sigma]_\varepsilon$ is a path from the root to one leaf in T_S
So, we have $[\omega]_\varepsilon \in T_S$

□

Proposition 3

Let $\omega \in L(S)$ be a timed word and $[\omega]_\varepsilon \in T_S$ its digitization for $\varepsilon \in [0,1[$
If $\exists a \in Act$ and $\exists \tau' \in \mathbb{N}$ such that $[\omega]_\varepsilon . a\tau' \in T_S$ then $\forall \tau \in]\tau' - 1 + \varepsilon, \tau' + \varepsilon]$ we have $\omega . a\tau \in L(S)$

Proof

Let $\omega = a_1\tau_1 a_2\tau_2 \dots a_n\tau_n$ be a timed word and
$[\omega]_\varepsilon = a_1[\tau_1]_\varepsilon a_2[\tau_2]_\varepsilon \dots a_n[\tau_n]_\varepsilon$ its digitization for $\varepsilon \in [0,1[$
We suppose that there exists $a \in Act$ and $\tau' \in \mathbb{N}$ such that $[\omega]_\varepsilon . a\tau' = a_1[\tau_1]_\varepsilon a_2[\tau_2]_\varepsilon \dots a_n[\tau_n]_\varepsilon a\tau'$ is a timed word of T_S.
From the definition of timed words, there exists a computation sequence $\sigma' = (q_0, v_0^\varepsilon, [\tau_0]_\varepsilon) \rightsquigarrow (q_1, v_1^\varepsilon, [\tau_1]_\varepsilon) \rightsquigarrow \cdots \rightsquigarrow (q_n, v_n^\varepsilon, [\tau_n]_\varepsilon) \rightsquigarrow (q, v', \tau')$ of T_S that

represents a path from the root to one leaf of T_S such that $a_i = \delta(q_{i-1}, q_i)$ for $i = 1, \ldots, n$ and $a = \delta(q_n, q)$
Proposition 1 ensures that σ' is the digitization of one computation sequence σ of S ($[\sigma]_\varepsilon = \sigma'$) with $\sigma = (q_0, v_0, \tau_0) \rightsquigarrow (q_1, v_1, \tau_1) \rightsquigarrow \cdots \rightsquigarrow (q_n, v_n, \tau_n) \rightsquigarrow (q, v, \tau)$ if we have $\forall x \in X, -\beta(x, \sigma) < v'(x) - v(x) < \beta(x, \sigma)$
We remember that from the digitization valuation definition, we have $\forall x \in X, v'(x) = v_n^\varepsilon(x) + \partial(q, x) * (\tau' - [\tau_n]_\varepsilon)$ and $v(x) = v_n(x) + \partial(q, x) * (\tau - \tau_n)$ which implies:

$$\Longrightarrow -\beta(x, \sigma) < v_n^\varepsilon(x) + \partial(q, x)(\tau' - [\tau_n]_\varepsilon) - v_n(x) - \partial(q, x)(\tau - \tau_n) < \beta(x, \sigma)$$

$$\Longrightarrow -\beta(x, \sigma) < v_n^\varepsilon(x) - v_n(x) + (\tau' - \tau) - ([\tau_n]_\varepsilon - \tau_n) < \beta(x, \sigma)$$

$$\Longrightarrow -\beta(x, \sigma) - (v_n^\varepsilon(x) - v_n(x)) + ([\tau_n]_\varepsilon - \tau_n) < \tau' - \tau < \beta(x, \sigma) - (v_n^\varepsilon(x) - v_n(x)) + ([\tau_n]_\varepsilon - \tau_n)$$

We remember that we have $-\beta(x, \sigma) < v_n^\varepsilon(x) - v_n(x) < \beta(x, \sigma)$ and $-1 + \varepsilon < [\tau_n]_\varepsilon - \tau_n \leq \varepsilon$
So we obtain $-\beta(x, \sigma) - \beta(x, \sigma) - 1 + \varepsilon < \tau' - \tau < \beta(x, \sigma) + \beta(x, \sigma) + \varepsilon$
In other terms, we have $\tau \in]\tau' - 1 + \varepsilon - \beta(x, \sigma) - \beta(x, \sigma), \tau' + \varepsilon + \beta(x, \sigma) + \beta(x, \sigma)$
From the digitization quantum definition, we have $\tau \in]\tau' - 1 + \varepsilon, \tau' + \varepsilon]$ which is included in the above interval.
We conclude that $\omega.a\tau \in L(S)\ \forall \tau \in]\tau' - 1 + \varepsilon, \tau' + \varepsilon]$

□

Proposition 4

Let $\omega = a_1\tau_1 a_2\tau_2 \ldots a_n\tau_n$ be a timed word and $\varepsilon \in [0,1[$.
If ω corresponds to a path from the root to a leaf in T_S
Then $\exists \omega' \in L(S)$ such that $[\omega']_\varepsilon = \omega$

Proof

We proceed by a recursive proof on the size of ω.
Let $\omega_i = a_1\tau_1 a_2\tau_2 \ldots a_i\tau_i$ with $i \leq n$ be the timed word obtained in the level i of the test tree, we have $\omega_n = \omega$
For $i = 0$, $\omega_0 = \emptyset$ the proposition is true because $(\omega'_0 = \emptyset)\ \omega'_0 \in L(S)$ and we have $[\omega'_0]_\varepsilon = \omega_0$
For $i < n$ we suppose that the proposition is true for i and we try to demonstrate for i+1
$\exists\ \omega'_i \in L(S)$ such that $[\omega'_i]_\varepsilon = \omega_i$
Given $a\tau$ such that we have $\omega_i.a\tau \in T_S$
From the proposition 3, $\forall \tau' \in]\tau - 1 + \varepsilon, \tau + \varepsilon]$ we have $\omega'_i.a\tau' \in L(S)$
So $[\omega'_i.a\tau']_\varepsilon = \omega_i.a\tau$

□

Now, we present the theorem that demonstrates that a timed word corresponds to a path in the test tree from the root to one leaf if and only if there exists a timed word in the specification model such that the path in the test tree corresponds to its

digitization. In other terms, for every timed word belonging to the specification model, their all possible digitizations for $\varepsilon \in [0,1[$ belong to the test tree.

Theorem

Let $\omega = a_1\tau_1 a_2\tau_2 \ldots a_n\tau_n$ be a timed word and $\varepsilon \in [0,1[$.
ω corresponds to a path from the root to a leaf in T_S
If and only $\exists \omega' \in L(S)$ such that $[\omega']_\varepsilon = \omega$

Proof

The proof of this theorem is a deduction of both proposition 2 and 4.

□

This theorem is important, because the test tree contains all digitizations of all timed words of the specification model. So, we ca n generate test cases by considering discrete time.

6 Conclusion

We presented a formal framework for generating test cases for duration systems. First, we described the specification of a duration system by a duration variables timed graphs. We used the digitization technique in order to reduce the infinite set of states to a countable set of states. The digitization technique associated to every real value taken by a duration variable an integer value. We demonstrated that the absolute of the difference between the real value taken by a duration variable and its digitization is bounded by the number of restarts of the duration variable. Thanks to this result, we derived directly the test cases from the specification model by considering discrete time. We presented those test cases with a tree.

Second, we demonstrated that a timed word ω is path in the test tree if and only if there exists a timed word ω'in the specification model such that ω is its digitization.

In the future works, we intend to extend our framework to nondeterministic duration systems. Another work consists ton investigate on the coverage criteria. Also, other models can be considered such as hybrid systems.

References

[1] Alur, R., Courcoubetis, C., Dill, D.: Model-Checking for Real-Time Systems. In: 5th Symp. On Logic in Computer Science. IEEE, Los Alamitos (1990)

[2] Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)

[3] Bouajjani, A., Echahed, R., Robbana, R.: Verifying Invariance Properties of Timed Systems with Duration Variables. In: Proc. Formal Techniques in Real-Time and Fault Tolerant Systems, FTRTFTS 1994 (1994)

[4] Briones, L.B.: Theories for model-based testing: real-time and coverage, thesis. University of Twenty (March 2007)

[5] Cerans, K.: Decidability of Bisimulation Equivalence for Parallel timer Processes. In: CAV 1992. LNCS, vol. 663. Springer, Heidelberg (1992)

[6] Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, p. 138. Springer, Heidelberg (2000)

[7] En-Nouaary, A., Dssouli, R., Khendek, F., Elqortobi, A.: Timed Test Cases Generation Based on State Characterization Technique. In: RTSS 1998. IEEE, Los Alamitos (1998)

[8] Henzinger, T., Manna, Z., Pnuelli, A.: What good are digital clocks? In: ICALP 1992. LNCS, vol. 623 (1992)

[9] Hessel, A., Petterson, P.: A Test Case Generation Algorithm for Real-Time Systems. In: Proc. of the 4th International Conference on Quality software, QSIC 2004, pp. 268–273 (2004)

[10] Khoumsi, A., Jéron, T., Marchand, H.: Test Cases Generation for Nondeterministic Real-Time Systems. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 131–146. Springer, Heidelberg (2004)

[11] Krichen, M., Tripakis, S.: Black-Box Conformance Testing for Real-Time Systems. In: SPIN 2004 Workshop on Model Checking Software (2004)

[12] Majdoub, L., Robbana, R.: Testing Duration Systems using an Approximation Method. In: Second International Conference on Dependability of Computer Systems Depcos-RELCOMEX 2007, Szklarska Poreba, Poland, June 207, pp. 119–126 (2007)

[13] Majdoub, L., Robbana, R.: Testing Duration Systems. Journal Européen des Systèmes Automatisés, approches formelles pour la validation de systèmes temps-réel 42(9), 1111–1134 (2008)

[14] Majdoub, L., Robbana, R.: Soundness Test cases Generation for Duration Systems. In: Proc. third International Design and Test Workshop, pp. 57-62, Monastir, Tunisie (December 2008)

[15] Majdoub, L., Robbana, R.: Test Cases Generation for Nondeterministic Duration Systems. In: 7th International Workshop on Modeling, Simulation, Verification and Validation of Enterprise Information Systems- MSVVEIS 2009, Milan, Italy, May 2009, pp. 14–23 (2009)

[16] Robbana, R.: Verification of Duration Systems using an Approximation Approach. Journal of computer Science and Technology 18(2), 153–162 (2003)

[17] Tretmans, J.: Testing Concurrent Systems: A Formal Approach. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 46–65. Springer, Heidelberg (1999)

Dynamic Model Initialization Using UML

Lila Meziani and Thouraya Bouabana-Tebibel

Ecole Nationale Supérieure d'Informatique, LCSI loratory,
BP 68M Oued-Smar 16309 Algiers Algeria
e-mail: {l_meziani,t_tebibel}@esi.dz

Abstract. UML Formalization is often undertaken by projecting the notation in a rigorously defined semantic domain. When the target formalism is of state transition type, the derived models are verified by model checking. The checking is performed on an accessibility graph generated from an initialized dynamic model. We propose, in this paper, an approach to initialize object Petri nets models at a time t different from the initial time. Object Petri nets are derived from sate machines. They are initialized using object diagrams. The approach associates the object state specified on the object diagram to the one specified on the state machine. A case study is given to illustrate the approach.

1 Introduction

Notations based on diagrams such as UML [17] are classified as semi-formal. Many works [5,6,10,11,14,20,21] study the semantics of the UML proposing to give it more precision. This precision is usually obtained through a denotational semantics projecting the notation in a rigorously defined semantic domain. The formal models, derived from the UML diagrams, may afterwards be verified by model checking. The checking consists in verifying that the derived formal model satisfies safety and liveness properties. Verification is performed on the reachability graph which is generated to propose all execution alternatives starting from an initial marking.

In UML, data initialization is provided by means of object diagrams specifying the objects and messages identity, their attributes as well as their state at the time of initialization. Objects state can be omitted when, by default, all data are given for the initial state of the system life cycle. However, in software engineering, some systems need sometimes to be studied beginning from a state that is different from their initial state. Indeed, when a system already exits, and designers only project to update, restructure or extend its functionalities, just a part of its life cycle is revised or added. In these cases, some objects of the system are already in action, and so, are localized on states that are necessarily different from the initial one. This reduces the accessibility graph which will be truncated of all the states space preceding the new system starting. Another advantage behind starting a system behavior at a time different from the initial one consists in reducing the states space and consequently allowing the system validation by model checking.

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 169–178.
springerlink.com

Our interest in this work is in the initialization of objects Petri nets (OPNs) models, derived from state machines, at a time t different from the initial time. State machines are noted in the following SM. The objects distribution on the model, at the appropriate time, will be deduced from the object diagram. The latter is, precisely, introduced to define the objects sate at a given time, and to specify their identity and attributes value. The object state is written in the object identity compartment. It will be used, in conjunction with the state machine, to locate the object on Petri nets.

This paper begins with a brief presentation of the state machines formalization work, published in [4]. In section 3, the proposed approach is presented and techniques on which it rests are developed. We provide in section 4 the reasons that motivate our work and show its novelty and relevance by comparison with related works. Section 5 deals with the approach validation. We conclude with some observations on the obtained results and recommendations for future research directions.

2 Related Works

Some studies have already addressed the formalization of UML dynamic diagrams by translating them into OPNs semantics domain. The most known are those of Baresi. He begins in [1] by a textual and graphical formalization of dynamic diagrams through the OPNs. He afterwards proposes in [2] translation recommendations. He only achieves the formulation of formal conversion rules for syntactic models in [3]. The drawback of this proposal is the constraint to write the UML models in a canonical language called LEMMA. Bokhari and Poehlman offer in [7] to transform UML state machines in OPNs in order to analyze them. The model validation resulting from the derivation is performed on the model checker DesignCPN. No details are however given about the initialization of the model that deals with identified objects.

Similarly, Hsiung et al. presented in [12] an approach for the formalization of statecharts with colored Petri nets. For this purpose, they use sequence diagrams to initialize their models and OCL constraints transformed into temporal logic to validate them. But the model initialization starts from time zero.

Fish and Störrle offer in [9] a number of principles, applicable to visual languages, characterized by imprecise semantics in order to analyze and discuss their quality. Based on this approach, they identify many sources of potential errors in UML diagrams and propose solutions to these deficiencies.

New approaches of the UML formalization techniques are graph transformation [11] and more recently, grammar graphs [15]. These techniques give more precision to the UML diagrams semantics without the use of formal languages.

3 Background

We present in this section the main results of the state machine transformation approach into OPN. We developed this approach in [4].

To illustrate its mechanisms and those proposed in this paper, we take a case study on a centralized peer to peer system. The main activity of this system is the information exchange between the peers once identified by the server. Identification is established, after a connection request, granted by the server. Once connected, peers interact by exchanging information. Fig. 1 shows the class diagram for the application, illustrating the system objects and their actions.

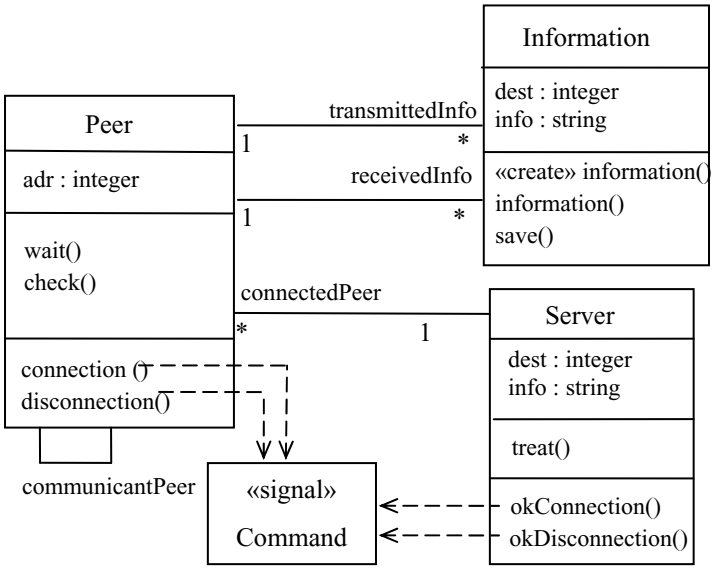


Fig. 1 Peer to peer class diagram

3.1 Transforming State Machines into OPNs

A state diagram [17] formally describes the behavior of objects of a given class, through states, when they receive or generate events. The generated events appear either on transitions or at the input or exit of states. They are noted on Fig. 2 by *evt*. The received events appear on transitions. They are noted *trg*.

In the OPN approach, classes are represented by subnets that can be instantiated as many times as needed to describe, in a nominative manner, the objects dynamics. This instantiation is done using tokens, written in the form of n-tuples, to model class instances. According to the object-oriented concepts, the subnet encapsulates the attributes and class methods. The attributes are expressed as components of the n-tuple (token) representing the object. Concerning the methods, they are specified in a flow of places, transitions and functions describing the

object life cycle. Places are categorized into simple places and super places. The simple places are those defined for ordinary Petri nets [13]. They include single tokens. The super places generate these tokens. Transitions are also of two types: simple and super. A simple transition models a single action. The super transition represents an internal processing described by a set of actions. Transitions can be guarded.

We proposed in [4] to specify the semantics of state machines by means of OPNs. The mapping results are represented in Fig. 2. Thus, each diagram SM is derived into a diagram called OPN, see Fig. 3. All the derived state machines SMs, related to classes of the system, are then connected by the *link* place.

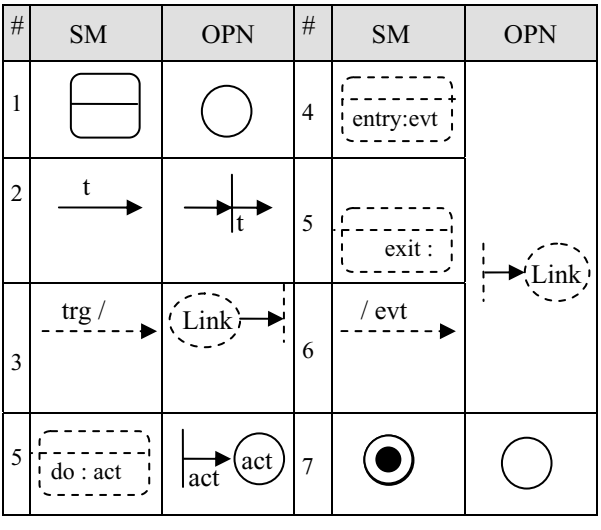


Fig. 2 Transformation of SM constructs into OPN

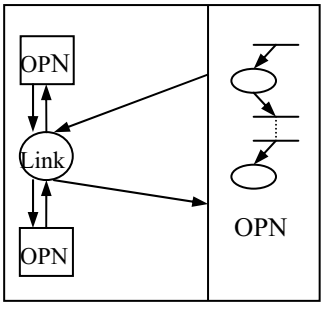


Fig. 3 OPNs interconnection architecture

3.2 Value-Oriented Specification

Validation of OPNs is established on the basis of their consistency with the system properties, transcribed in temporal logic. However, the formulation of system properties in temporal logic can be a hard task for the UML designer, unfamiliar with this type of formalism. The ideal, in this case, will be to give him the possibility to develop these properties in a familiar language. OCL, Object Constraint Language [18], seems to be the appropriate formalism. It is an integral part of the UML notation for expressing constraints on models while conserving their readability.

The expressions of OCL invariants are mostly based on the manipulation of collections. As these collections correspond to association ends, the movement of objects that visit them must appear on the dynamic models so that the system properties can be checked on. This modeling is not usual when constructing dynamic models. For example, to model the connection of a peer to the server, the connection request and confirmation are always specified but the insertion of the connected station into the association end *Connectedpeer* is generally not mentioned. To overcome this, we require the modeling of object insertions and deletions at the association ends. This movement can be provided by the link actions, defined by the OMG work on the semantics of the UML actions [16]. These actions relate to the link creation and the link destruction as well as the destruction of all links of an association. Fig. 4 shows the state machine of a peer with some link actions specified on the association ends.

Our work on the integration of the objects movement through the association ends is presented in [5].

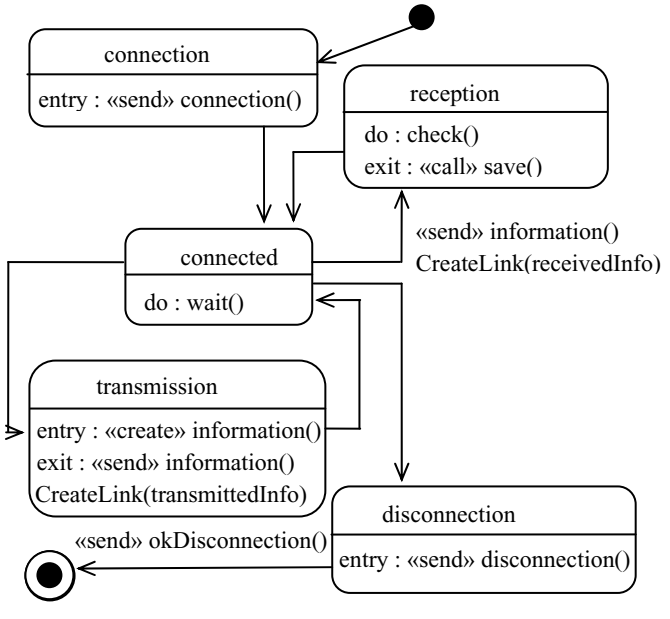


Fig. 4 State machine of a peer

4 Initialization Approach

The verification of OPNs models, derived from state machines, requires the initialization of the specification. Most of the research works [8,19,21] undertake this validation with an initial marking made of anonymous objects. Such marking is appropriate when one has to evaluate particularly the objects dynamics characteristic. When the interactivity feature is taken into account, the verification with anonymous objects proves to be insufficient because it inhibits many aspects of the communication. Indeed, running the verification by considering a single object as class representative may remove any meaning to inter-classes communication, especially when anonymity is on the exchanged messages.

To remedy this, we initialize the marking of OPNs models by considering objects and messages, identified by names and attribute values. Thus, the object is identified by the 2-tuple <*obj, attrib*> where *obj* is its identity and *attrib* its attribute values. The message will be identified by the 3-tuple <*assoc, obj, attrib*> where *assoc* designates the identity of the object to which it is associated. The initialization is deduced from object and sequence diagrams.

4.1 Syntax and Semantics of Object Diagrams

The object diagram [17], also called instances diagram, shows the structural links between class instances at a given time. It thus constitutes the system structural state at a precise moment. It is composed of objects symbolized by rectangles with two compartments. The first compartment contains the instance name concatenated to the one of the class as follows: Class:object. The state of the object may be specified in brackets. It corresponds to the object state on the state machine at a given time. Thus, the system state at a given time is provided by the objects state specified into the first compartment. In the second compartment, the attributes of the object are initialized with values. The associations between objects show the links between these objects at a given time. Thus, at time zero of a system life, no link associates the class instances. Otherwise links exist with names on their ends, see Fig. 5.

More formally, an object diagram can be specified by the n-tuple OD= <*Cl, O,* S_{DO}*, Atr, L, R, Inst, Ext, Stat, St, At,* O_{in}*,* O_{ass}> where:

- *Cl = {cl} is a set of instantiated classes on the diagram,*
- *O = {o} is a set of objects of the diagram,*
- S_{DO} *= {s} is a set of states of objects on the diagram,*
- *Atr = {attrib} is a set of the objects attributes,*
- *L = {l} is a set of links,*
- *R = {r} is a set of association ends,*
- *Inst: Cl → P(O) is a function that returns the class instances,*
- *Ext: Cl → P(R) is a function that returns the association ends,*
- *Stat: Cl → P(S) is a function that returns the object state in the object diagram,*
- *St: O → S is a function that returns the state of an object,*

- *At: $O \rightarrow P(Atr)$ is a function that returns the object attributes,*
- *$O_{in} : R \rightarrow O$ is a function that returns the object at the association end,*
- *$O_{ass} : R \rightarrow O$ is a function that returns for an association end, the object at the opposite end.*

4.2 Distribution of Objects on the OPNs

The simulation of an OPN model, starting from any state of its lifecycle, requires an appropriate placement of its objects into the corresponding places. These places, which derive from the state machine diagram states, also correspond to the states specified within the object compartments of the object diagram, see Fig. 5. Thus, each object modeled in the object diagram is positioned on the OPN modeling its dynamics, at the place corresponding to its state specified on the object

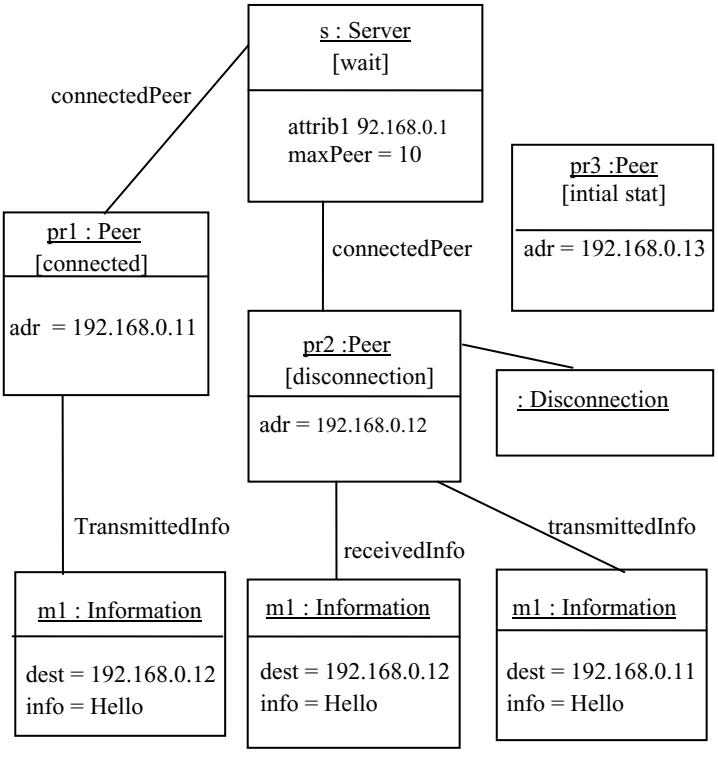


Fig. 5 Object diagram of the system at time t

diagram. In addition to the OPN places, those related to the association ends must also be initialized. Thus, each place representing an association end receives all objects of the association end specified on the object diagram. More formally:

Function marqObjectt (OD)
- *for each class cl ∈ Cl :*
  - *for each state s ∈ Stat(cl) :*
    - *for each object obj∈ Inst(cl) such that St (obj) = s:*
      - *create col = <obj, attrib>, attrib = At (obj)*
      - *set* $M_0(p) = \sum_{col} <obj, attrib>$, $p = D_e(s)$,
- *for each association end r ∈ Ext(cl) :*
  - *for each link l ∈ L such that l = r :*
    - *create col = <assoc, obj, attrib>, assoc =* $O_{ass}(r)$,
      obj = $O_{in}(r)$, *attrib = At (obj),*
    - *set* $M_0(rol) = \sum_{col} <assoc, obj, attrib>$, $rol = D_e(r)$

where $M_0(p)$ is the marking of place p and *assoc* is the class to which the object *obj* is associated.

To illustrate this concept, we propose the object diagram of Fig. 5 where the server s is already connected to *pr1* and *pr2* peers. The peer *pr3* is not yet connected. The peer *pr1* is in a *connected* state after it has sent the message *m1*. The peer *pr2* has received the message *m1*, answered it by the message *m2* and then placed itself in a *disconnected* state.

5 Validation of the Approach

To test the proposed approach, we built a tool whose components work as follows. We first developed a graphic interface to construct the used UML diagrams, namely state machines and object diagrams. We afterwards, implemented a translator which derives OPNs from state machines. The derived OPNs were proved to be well constructed and faithful to the client requirements. This checking was performerd by means of the model checker PROD [22].

Verification by model checking, as treated in PROD, is based on the state space generation and the verification of safety and liveness system properties on this space. The properties may be basic, about the correctness of the model construction or specific, written by the modeler to ensure the faithfulness of the system modeling. For each of these approaches, given a property, a positive or negative reply is obtained. If the property is not satisfied, it generates a trace showing a case where it fails.

The basic properties are verified according to two ways: the on-the-fly tester approach and the reachability graph inspection approach. *The on-the-fly tester approach* detects deadlock, livelock and reject states. As for *the reachability inspection approach*, it permits the verification of some other properties such as quasi-liveness, boundedness or reinitializability. For a more precise validation, specific properties of the system can be written by the designer in Linear temporal Logic (LTL) or Computational Tree Logic (CTL) and then, verified by PROD.

Once the OPNs generated and verified, we used the object diagrams to initialize them. This was performed using the implemented algorithm MarkObject(OD). The obtained marking reveals to be conform to the object model.

6 Conclusion and Perspective

The Many research results are published on the formalization of the UML but none so far on the initialization of the derived formal models, starting from object and sequence diagrams, established at times different from the time zero. This paper proposes an approach to validate models, derived from the state machines, at any time of the system life cycle. The initialization of these models is obtained from object diagrams. The OPN model initialization also includes places representing association ends. This specification allows a better model validation using OCL constraints.

The proposed solution totally rests on UML diagrams taking advantage of a construct already available on the object diagrams, namely the object state. However, it requires from the UML modeler to specify the state of each object modeled in the object diagram.

An interesting perspective to this work is to perform OPNs model initialization without resorting to the states specified on the object diagrams. The use of the association ends, modeled on those diagrams, is a promising research direction.

References

[1] Baresi, L., Pezzé, M.: On Formalizing UML with High-Level Petri Nets. In: Agha, G., De Cindio, F., Rozenberg, G. (eds.) APN 2001. LNCS, vol. 2001, pp. 276–304. Springer, Heidelberg (2001)

[2] Baresi, L.: Some Premilinary Hints on Formalizing UML with Object Petri Nets. In: Proc: 6th World Conference on Integrated Design & Process Technology, Pasadena, USA (2002)

[3] Baresi, L., Pezzè, M.: Formal interpreters for diagram notations. ACM Trans. Softw. Eng. Methodol. 14(1), 42–84 (2005)

[4] Bouabana-Tebibel, T.: Object dynamics formalization using object flows within UML state machines. Entreprise Modelling and Information Systems Architectures 2(1), 26–39 (2007)

[5] Bouabana-Tebibel, T.: Roles at the basis of UML validation. Journal of Computing and Information Technology 15(2), 171–183 (2007)

[6] Bouabana-Tebibel, T., Belmesk, M.: An Object-Oriented approach to formally analyze the UML 2.0 activity partitions. Information and Software Technology 49(9-10), 999–1016 (2007)

[7] Bokhari, A., Poehlman, W.P.S.: Translation of UML Models to Object Coloured Petri Nets with a view to Analysis. In: SEKE 2006, pp. 568–571 (2006)

[8] Delatour, J., De Lamotte, F.: ArgoPN: A CASE Tool Merging UML and Petri Nets. In: Proc: 1st International Workshop on Validation and Verification of software for Enterprise Information Systems, ICEIS, Angers (2003)

[9] Fish, A., Störrle, H.: Visual qualities of the Unified Modeling Language:Deficiencies and Improvements. In: IEEE Symposium on Visual Languages and Human-Centric Computing - VL/HCC 2007, pp. 41–49 (2007)

[10] Harel, D., Maoz, S.: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. In: Proc. 5th Int. Wsh. Scenarios and State Machines: Models, Algorithms, and Tools - SCESM 2006, pp. 13–20. ACM Press, New York (2006)

[11] Holscher, K., Ziemann, P., Gogolla, M.: On translating UML models into graph transformation systems. Journal of Visual Languages and Computing 17, 78–105 (2006)
[12] Hsiung, P.-A., Lin, S.-W., Tseng, C.-H., Lee, T.-Y., Fu, J.-M., See, W.-B.: VERTAF: an Application Framework for the Design and Verification of Embedded Real-Time Software. IEEE Transactions on Software Engineering 30(10), 656–674 (2004)
[13] Jensen, K.: An Introduction to the Practical Use of Coloured Petri Nets. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1492, pp. 237–292. Springer, Heidelberg (1998)
[14] Knapp, A., Wuttke, J.: Model Checking of UML 2.0 Interactions. In: Auletta, V. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 42–51. Springer, Heidelberg (2007)
[15] Kong, J., Zhan, K., Dong, J., Xu, D.: Specifying behavioral semantics of UML diagrams through graph transformations. The J. of Syst. and Soft. 82, 292–306 (2009)
[16] Object Management Group, The UML Action Semantics (2001)
[17] Object Management Group, UML 2.0 OCL Specification (2003)
[18] Object Management Group, UML 2.0 Superstructure Specification (2004)
[19] Saldhana, J.A., Shatz, S.M., Hu, Z.: Formalization of Object Behavior and Interactions From UML Models. International Journal of Software Engineering and Knowledge Engineering – IJSEKE 11(6), 643–673 (2001)
[20] Staines, T.S.: Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets. In: 15thIEEE Int. Conf. and Workshop on the Engineering of Computer Based Systems, pp. 191–200. IEEE Xplore, Belfast (2008)
[21] Störrle, H., Hausmann, J.H.: semantics of uml 2.0 activities. Software Engineering, 117–128 (2005)
[22] PROD 3.4, An advanced tool for efficient reachability analysis. Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland (2004)

Integrated Application of Compositional and Behavioural Safety Analysis

Septavera Sharvia and Yiannis Papadopoulos

University of Hull,
Cottingham Road, HU6 7RX, Hull, UK
email: {s.sharvia,y.i.papadopoulos}@hull.ac.uk

Abstract. The design complexity of modern safety critical systems presents various challenges for its safety assessment process. In recent years, Model-Based Safety Analysis (MBSA) has been proposed to achieve more-robust and effective safety assessment techniques through automation of the synthesis and analysis of predictive models. Two prominent paradigms of MBSA are Compositional Safety Analysis (CSA) and Behavioural Safety Analysis (BSA). These techniques have emerged with little integration. In this chapter, we present a technique which systematically integrates the application of CSA and BSA. The process starts from CSA and utilizes its analysis results to provide a systematic construction and refinement of state machines, which can be subsequently analyzed through BSA. An example of a car brake-by-wire system is presented to illustrate the application of the proposed technique.

1 Introduction

Safety critical systems are systems whose operational deviations can potentially lead to catastrophic consequences or loss of human lives. These systems are widely employed in various industries, including the automotive, aerospace, weapons and nuclear industries. Today's modern safety-critical systems often incorporate numerous embedded control components, involve various engineering disciplines, and employ distributed architectures and complex communication structures. In such systems, achieving design solutions that fulfill safety requirements remains a challenge. Classical safety analysis techniques such as Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) are popular techniques employed to predict the safety of such systems. However, these techniques are traditionally manual, and in the context of a complex system become difficult, laborious, expensive and error-prone.

This limitation results in the FTA and FMEA performed only at the later stage of lifecycle when the design has been finalized. This late contribution means that results from the process miss the opportunity to influence system design, which could incur extra cost and effort in late design modifications. Challenges also arise in the lack of systematic methods to capture and manage design models and safety artefacts as in traditional practices, system design models and safety analyses are often handled separately. With these drawbacks, classical safety analysis

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 179–192.
springerlink.com

techniques face tremendous challenges and are no longer deemed to be sufficiently effective and robust in managing the rising intricacy of modern complex design.

Model-Based Safety Analysis (MBSA) has been proposed in the recent years to address some of these problems. Focus has been placed on developing more-effective and robust safety assessment techniques through automation of the synthesis and analysis process. MBSA introduces semi-formal and formal models in the centre of the design and assessment process. Effort is focused on the construction of the formal specification of the system model. This specification model is subsequently used as the foundation for various development activities like prototyping and simulation, code generation, and testing [9]. To perform a thorough safety assessment, it is crucial to understand not only how a system behaves in its normal working condition (represented in the *nominal* model), but also in the presence of failure(s). This is done by extending the nominal model with failure information to construct the failure-augmented model, termed *fault model* [10] or *error model* [13].

Automated analysis of models brings substantial benefits as it simplifies the process, lightens the burden on designers and analysts, saves time and contributes to more reliable results. It also enables safety analysis to be incorporated as part of an iterative design process - as new results can be more easily generated to reflect changes – and therefore driving the design with safety in mind.

The two most prominent paradigms of MBSA today are *Compositional Safety Analysis (CSA)* and *Behavioural Safety Analysis (BSA)*.

CSA techniques are widely used to assist the process of reliability engineering. It uses predictive models of system failure which can be produced in the form of well-known safety artefacts like fault trees. System models can be decomposed into structural hierarchies, and the local failure logic of components in these hierarchies is provided by analysts. Faults trees or FMEAs are then automatically produced by establishing how the local effects of component failures combine as they propagate through the topology of the system. The process is flexible and adaptable to different stages of model development. This is particularly valuable as assessment can be started early in the design process when concrete system details are still minimal. CSA produces safety artefacts (e.g. fault trees) which are familiar to safety engineers, making the process more intuitive. These artefacts identify potential failures and design weaknesses which can guide possible design modifications, and help to derive and refine requirements. CSA techniques allow quantitative analysis and in some cases also architectural optimization. Despite these strengths, CSA techniques provide no support for formal verification. Analyses with FTA and FMEA are generally static, and do not take into consideration the changes in system states and are therefore unable to capture dynamic behaviour. This limitation has been to some extent addressed in HiP-HOPS with a recent extension that enables assessment of sequences of failures via synthesis of temporal fault trees and FMEAs [14]. Techniques which are based upon the CSA approach include Hierarchically Performed Hazards Origin and Propagation Studies (HiP-HOPS) [1], Component Fault Trees [11], and State-Event Fault trees (SEFT) [7].

BSA, on the other hand, generally aims to facilitate system verification. In BSA, system-level effects of failures are constructed by injecting faults into the formal specification of the system. This technique commonly employs model checking to allow formal verification. Model checking verifies safety properties which represent safety requirements and enables the assessment of dynamic behaviour. Formal models are expressed as *state automata* (or "*finite state machines*") in the language of the particular technique, while safety properties are usually expressed in temporal logic. Model checker tool performs exhaustive exploration to assess whether a safety property holds for the system. The tool produces a counterexample when a safety property does not hold to show traces of 'simulation' on how the breaching condition is reached.

BSA facilitates automated formal verification and captures dynamic behaviours. It is also possible to differentiate between transient and permanent failures and model the temporal ordering of failures. However, its limitation include the fact that most model checker tools require the system model to be expressed in that particular model checker input language. Safety artefacts like fault trees produced from model checker generally have 'flat' structure representing disjunction of all minimal cut sets, which can hamper understanding of the fault trees. The analysis is also typically qualitative in nature, and not probabilistic. Formal models (which are required as the input to model checker) are only developed at later stage where designs are more mature, detailed and stable. Lastly, model-checking-based approaches are computationally expensive and inductive in nature, which means that the exhaustive assessment of the effects of combinations of component failures can potentially be infeasible in larger systems. Examples of techniques which are based on this approach include Altarica [2] and FSAP/NuSMV [3].

2 Integrated Application of Compositional and Behavioural Safety Analysis (IACoB)

CSA and BSA emerged with little integration. While the differences in strengths, limitations, and assessment objectives of both techniques are acknowledged, we also recognize that the above techniques are complementary and could yield substantial benefits when applied together. Here we propose a method called Integrated Application of Compositional and Behavioural Safety Analysis (IACoB). IACoB is a safety-driven method which exploits analysis results from quick iterative CSA techniques to derive a more-robust, safety-driven model prior to the application of BSA. The method can be iterated until a satisfactory design that fulfils safety requirement criteria is reached. HiP-HOPS is selected to facilitate CSA in IACoB as a representative CSA technique and the NuSMV model checker is selected to facilitate BSA.

IACoB starts with the construction of a system model, which can be an early functional model or a more detailed architectural model, depending on the stage of the system development. At early stages of design, a functional model of the system is established, which shows input, processing and output functions and

dependencies among them, e.g. the data exchanged among functions (or material and energy in the general case). Once the model is constructed, design engineers examine further this model in order to evaluate the severity of failures of output functions, i.e. functions provided by the system to users and its environment. Each function is then annotated with its local failure behaviour in the style of HiP-HOPS, enabling automated preliminary FTA to be conducted via application of CSA. This can be done by assigning each component a set of output deviations, input deviations and hypothesised internal malfunctions that lead to those output deviations. This allows failure logic to be developed and the propagation of failure to be established. Automated algorithms such as those in HiP-HOPS can then be applied to perform the automated construction and analysis of system fault trees and FMEAs. These analyses will show the effect of hypothesised component failure modes on system outputs, which in turn allows the results to be checked against safety requirements. By identifying and studying how a component failure might lead to a catastrophic system failure, safety measures can be devised, for example, by enforcing a requirement on the design of the component, modifying the system structure or introducing safety mechanisms and fault tolerant features. Focus is placed especially on component failures that may have catastrophic or critical effects on the system, as they need to be prevented by design - or at least their impact minimized. In general, FMEAs can be used to show the effects of hypothesised component failures, and then help decide whether failures can be tolerated or not. Intolerable failures must become sufficiently unlikely by appropriate design of the component or external detection and recovery mechanisms must be put in place. Tolerable failures may be allowed to happen resulting in loss of functionality but always in the context of a graceful transition to degraded-but-safe modes of operation. In the cases of tolerable failures, FMEA result leads to design solutions that introduce, or enforce, these new "degraded modes" where components have failed but the system maintains its safety related functions.

The results of the FMEA can therefore be used to drive the refinement of the design of components and system modes. The process of design refinement is systematic and guided by the results of CSA. In the next step of the process, abstract state machines are constructed to show how graceful transition to the identified degraded modes could be achieved. Driven by these results, design iteration take places to incorporate these new degraded modes in an improved version of the system model. In general, system states are organized into *functional states*, where on each state, the system delivers a different set of functions. Component failures typically cause functional losses and malfunctions which lead the system from normal to degraded and ultimately failed modes. Once state machines showing these transitions have been derived, model checking technology is used to verify that safety properties hold on these state machines and get assurance that early designs incorporate correct interpretations and specifications of safety requirements. BSA in the form of model checking can be used to verify both the general dynamic behaviour of the system and the particular effect of any fault tolerance mechanisms that have been introduced following interpretation of CSA results.

As the design of the system is refined, state machines can provide very detailed representations of behaviour. In the course of the proposed process, CSA and BSA are usefully being applied together. The results of the CSA help to improve the architecture of the system, via introduction of fault detection and fault tolerance where appropriate, but they also guide the construction of behavioural system models that can be subjected to rigorous and detailed BSA.

This process can be iterated as the design evolves. At later stages of design, CSA studies can become much more detailed and quantitative in nature. Failure annotations of components can be extended to include real failure modes and probabilistic component failure data. Such failure modes include electrical and mechanical of hardware caused by wear and environmental conditions or, in the case of composite systems that also include software, statistically observed functional failures caused by unspecified random or systematic faults. The results can be used to quantitatively predict the reliability, safety and availability of the system. Such prediction forms a necessary and important component of the system safety assessment process.

3 Example

3.1 Introduction to Brake-by-Wire System

Brake-by-wire systems replace traditional automotive braking components (like brake boosters, pumps, and master cylinders) with electronic sensors and actuators. Brake-by-wire systems can have hydraulic backup or not. The former, also known as Electric Hydraulic Brake (EHB) technology is realized through hydraulic pumps and additional electrically controlled valves. If the electronic control fails, the complete electric hydraulic system will be deactivated and the brake system will behave like a pure hydraulic system which delivers only emergency brake function with reduced brake force. Brake-by-wire without hydraulic backup is known as Electric Mechanical Brake (EMB) and uses only computer controlled electro-mechanical actuators. EMB does not possess the fail-safe mechanics of hydraulic backup, and therefore must be developed with strict fault tolerant properties.

The example brake-by-wire system used in this chapter is based upon a model described in [12] but also draws from designs in [8] and [4]. The system consists of one vehicle-level processing function and four local-level wheel processing functions. The vehicle-level processing function reads in brake command input from the driver, communicated through a human-machine interface (for example, the brake pedal or parking brake interface), and subsequently generates braking command for each local-level wheel processing functions based on high-level advanced brake functions such as an Anti-Lock Brake System (ABS). Local-level wheel processing functions are located physically close to the wheels and control the provision of braking energy. Upon receiving braking command from the vehicle-level processing function, each local-level processing function calculates the value of braking pressure, taking into consideration various local-level

information including actuator position and speed. This value of braking pressure is then fed to an actuator which then applies the actual braking pressure on the corresponding wheel of the car.

3.2 Analysis of System Functional Models

In accordance with the IACoB method, we start the safety assessment process from a high-level functional model. For this simplified system, two initial main functions can be delivered: 1) Function which delivers basic braking 2) Function which delivers braking with driving assistance anti-lock (ABS). These two functions can arguably be combined into one as they are not physically distinct. In this early model, however, they are free from architectural detail and are modelled as two separate logical functions to facilitate the illustration of function delivery.

The Matlab Simulink model illustrated in Fig. 1 represents a high-level abstraction of the brake-by-wire system. It is simplified to consist of input functions, braking command processing functions (vehicle-level and local-level), ABS command processing function, and output functions. As local-level processing provides identical function for each wheel of the vehicle, we assume it is sufficient to discuss and analyze one (instead of all four) in this initial model. There are four input blocks which read the driver demand from the brake pedal (Input_brakeDemand), readings for wheel speed (Input_wheelSpeed), external variable readings (Input_external) , and local-level feedback (Input_local). Information on brake demand, wheel speed and the external environment is passed to the vehicle-level processing function (VehicleLevelProcessing) which calculates and generates the independent brake commands for each local-level processing (LocalLevelProcessing). It also relays the information needed for ABS calculation to the ABSProcessing function. The wheel local-level processing controls the output functions which provide braking energy for basic braking or ABS braking. This early model does not yet incorporate any fault tolerance mechanisms.

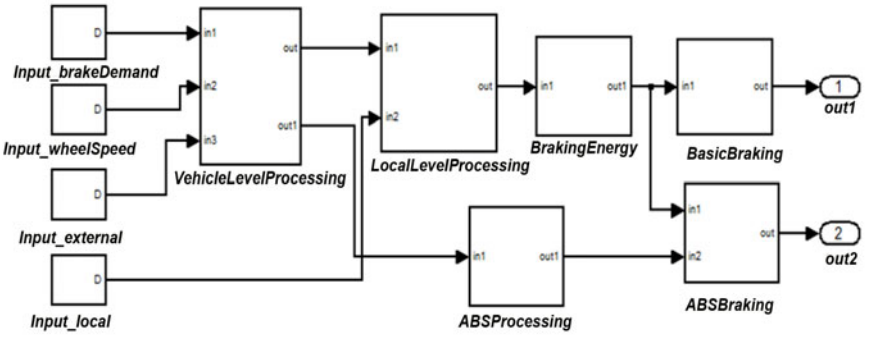


Fig. 1 Abstract functional model for brake-by-wire system

3.3 Functional Failure Analysis

Once the model is constructed, we proceed to perform the FFA on the system. The main aim of this process is to classify and analyse the effects and severity of failures in the output functions, BasicBraking and ABSBraking. In this case the focus is placed on the omission and commission failure types, although it is also possible to perform analysis on value or timing failures. Table 1 presents an extended FFA which includes identification of detection, potential recovery plan and recommendation columns for each failure.

From the examination of this FFA table, it can be seen that the severity of an omission failure of function BasicBraking (O-BasicBraking) is categorized as having a catastrophic effect, and therefore should be mitigated with fault tolerant design. The second functional failure related to the provision of braking pressure is commission. The commission failure in BasicBraking function (C-BasicBraking) is identified as having critical consequences and therefore should not be allowed to propagate and wrongly influence other functions. One way to achieve this is by detecting the commission failure, forcing the system to fail silent and then handling the omission accordingly by putting a fault-tolerant mechanism in place. The failure for the ABSBraking function is categorized as having catastrophic severity in its commission failure and marginal effects in its omission failure. This is due to the nature of the ABSBraking function which provides driving assistance rather than those of imperative role in braking. This suggests that it is more favourable for the function to fail in omission, and therefore the function should fail-silent when commission failure is detected.

Table 1 Functional failure analysis for brake-by-wire system

Function	Failure Type	Effects on System	Severity	Detection	Recovery Plan	Design Recommendation
BasicBraking	Omission	No brake force ; vehicle cannot be stopped; driver loses control.	Catastrophic	Using pressure feedback	Not possible	Redundant back up mechanism should be introduced
BasicBraking	Commission	Vehicle tends to drift; loss of stability	Critical	Comparing pedal input (demand) and pressure feedback	Release Pressure	Commission failure should not be allowed to propagate
ABSBraking	Omission	Loss of steerability; less efficient brake	Marginal	Using feedback on wheel speed and pressure	Not possible	Situation can be compensated by driver
ABSBraking	Commission	No brake force available	Catastrophic	Comparing wheel speed and pressure feedback	Switch off ABS function	Commission failure should not be allowed to propagate

3.4 FMEA

Functional blocks in the model are then annotated with failure behaviour before fault trees and FMEA can be generated and analyzed using HiP-HOPS tool. As the initial design does not include any fault-tolerant strategies, the initial FMEA table (Table 1) shows how each internal malfunction (recorded in "Failure Mode" column) in every function can become direct contributors to the omission and commission failures of the braking and ABS functions.

To implement a more robust design, several advisable design changes can also be determined from an analysis of the FMEA table. Severity is the effect of functional failure on the output functions of the system. Based on this effect, the criticality of the functional failure can be established and a recommendation can be made. One important (and most obvious) technique to achieve fault-tolerance is the introduction of redundancy in the 'module'. Module here refers functions in functional models or components in more refined architectural models. In common practice, fault tolerant design for brake-by-wire systems can be implemented through either the inclusion of a hydraulic system (EHB system) or through replicated electronic components (EMB system). For this example, we introduce a hybrid system which implements both hydraulic as well as redundant electronic modules (with lower numbers of redundant modules compared to a pure electronic EMB).

Table 2 Initial FMEA results for brake-by-wire system

Function	Failure Mode	Direct Effect	Severity	Comments /Recommendation
Input_brakeDemand	BDBE	O-BasicBraking	Catastrophic	Redundancy required
Input_external	ESBE	O-ABSBraking	Marginal	
Input_locals	LSBE	O-BasicBraking	Catastrophic	Redundancy required
Input_wheelSpeed	WSBE	O-ABSBraking	Marginal	-
VehicleLevelProcessing	VLPBEabs	O-ABSBraking	Marginal	-
	VLPBE	O-BasicBraking	Catastrophic	Redundancy required
	VLPBEc	C-BasicBraking	Critical	Should fail silent
	VLPBEabsC	C-ABSBraking	Catastrophic	VLPBE should not propagate and when detected, ABS should be deactivated.
LocalLevelProcessing	LLPBE	O-BasicBraking	Catastrophic	Redundancy required
	LLPBEc	C-BasicBraking	Critical	Should fail silent
BrakingEnergy	ActBE	O-BasicBraking	Catastrophic	Redundancy required
	ActBEc	C-BasicBraking	Critical	Should fail silent
ABSProcessing	ABSBE	O-ABSBraking	Marginal	-

The analysis of FMEA therefore provides an insight that assists us in distinguishing critical functional failures that contribute to failures which have catastrophic or critical consequences (O-BasicBraking, C-BasicBraking, C-ABSBraking)

from those that contribute to failures with marginal effects (O-ABSBraking). This knowledge subsequently allows us to establish the appropriate resource management priority and design improvement.

For example, Input_brakeDemand function and the Input_locals function are identified to be the contributing causes to O-BasicBraking which is catastrophic, and therefore it is necessary to configure these functions to be at least fail-operational by introducing a redundant module to backup each function. Failure in Input_external and Input_wheelSpeed only leads to O-ABSBraking and therefore in this example, will be tolerated. We also identified that there is a need to introduce a redundant function for VehicleLevelProcessing as its failure also leads to O-BasicBraking. Additionally, LocalLevelProcessing can be connected directly to the function Input_brakeDemand to read raw braking command. This way, when VehicleLevelProcessing function fails basic braking command can still be obtained. Similarly, an omission failure in basic braking caused by internal malfunction in LocalLevelProcessing and BrakingEnergy can be mitigated by introducing redundant functions to support these critical functions. In addition to this independent redundancy for individual modules, we could also include a hydraulic function which acts as the group backup mechanism to provide emergency braking in the presence of failures that affect the electrical-based functions. Commission failures on both braking and ABS functions have been identified as critical and catastrophic respectively. It is therefore recommended that any function which leads to commission failure should fail-silent instead. This can be achieved by deactivating or switching off the function whenever a commission failure is detected. This, in turn, transforms the commission failure into omission failure, which will then be treated accordingly. These changes are reflected in the revised model illustrated in Fig.2. Shaded areas within each module represent redundancy.

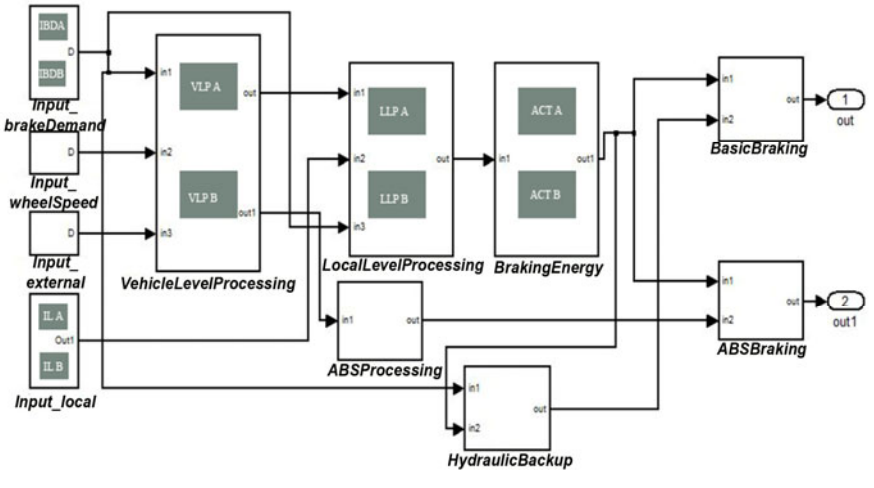


Fig. 2 Revised brake-by-wire system

The inclusion of these new redundant mechanisms results in the introduction of new failure behaviours, which requires the FTA and FMEA to be updated. The new fault-tolerant redundant structure means that there are no longer any single-point failures which directly cause O-BasicBraking.

3.5 Construction of State Machine

FTA and FMEA can be iterated until the design model meets early predefined requirements, for example until a satisfactory level of functional redundancy is achieved and the system is tolerant to n number of failures. In this example, we assume that elimination of single point failures for O-BasicBraking is sufficient. Next, we proceed and model the design dynamic behaviour by constructing an abstract state machine.

To do this, it is first of all, important to identify the primary elements of state machine: abstract states (referred to as 'modes') and transition events. Modes are derived based upon provision of system functions, which in this case are the BasicBraking function and the ABSBraking function. Here we summarize three modes the system that can be derived by considering the delivery of functions: 1) Normal (BBW_Normal) mode where both Braking and ABS functions are delivered. 2) Permanent Degraded (BBW_PD) mode where basic Braking is delivered, but ABS function can no longer be delivered. 3) Fail (BBW_Fail) mode where no braking pressure is delivered.

Transitions can be formulated according to the failures that could occur to each of the functions; in this case, all such failures are of omission type as commission failures have been transformed into omissions by design. To more closely reflect the inclusion of different type of pressure source, we could also refine the function BasicBraking into Electrical and Hydraulic. Subsequently, dynamic behaviour can now be modelled in the following modes:

1) Normal (BBW_Normal) mode where both basic braking and ABS braking functions are delivered. Braking function in normal mode is delivered through the primary source, Electrical module.

2) Permanent Degraded 1 (BBW_PD1) mode where braking function is delivered by the Electrical module, but the ABS braking function can no longer be delivered.

3) Permanent_Degraded2 (BBW_PD2) mode where braking pressure is delivered by Hydraulic module, ABS function is not delivered.

4) Fail mode where no braking pressure is delivered.

This can be illustrated in Fig. 3. It is also important to note how inclusive the transition definitions are when modelling dormant functions. For example, the mode chart in Fig. 3 is appropriate in a situation where the hydraulic back up is only activated when O-Electrical is detected. However, for 'hot standby' where hydraulic backup is continuously active, the transition definitions need to be updated. This is because it is possible for failure O-Hydraulic to occur when the system is operating in BBW_Normal mode.

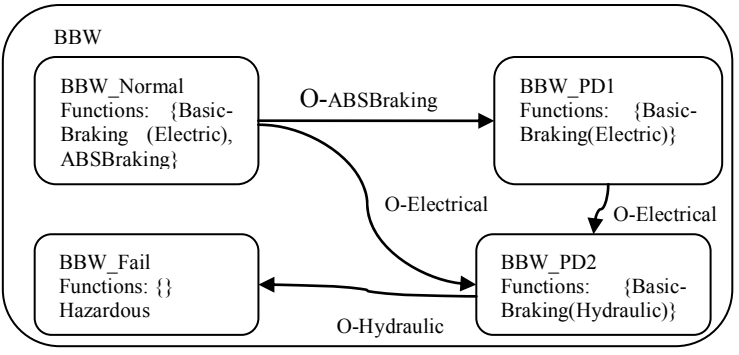


Fig. 3 Mode chart for brake-by-wire system

To promote a more transparent systematic degradation phase, it can be helpful to consider failure in hydraulic line during the normal operational mode. One possible way to better address this is by introducing an additional temporary mode (BBW_TD1), to model the failures in the Hydraulic function when basic braking is provided correctly through Electrical system. This degraded BBW_TD1 mode could serve as a potential warning that the backup function has failed before the primary function, a state in which potential recovery steps can also be included and performed. This can be shown in the following mode chart:

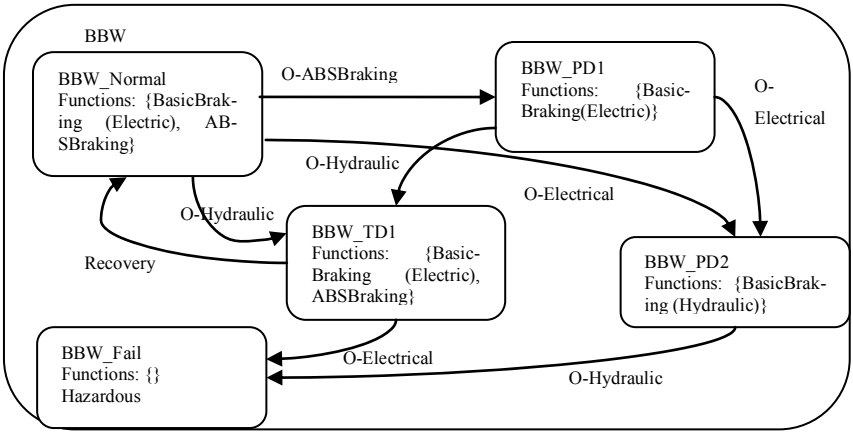


Fig. 4 Updated mode chart for brake-by-wire

The accuracy of safety assessment and verification of safety requirements depends on the level of detail provided in the mode chart. For this reason, it can be useful to refine the abstracted mode chart. This can be done by refinement of events through minimal cut sets or through compositional annotation. Refinement through minimal cut sets is performed by replacing failure events with its set of

minimal cut sets, while refinement through compositional annotation is done by utilizing HiP-HOPS failure annotation to establish connections between failure events.

3.6 Verification of Safety Requirements

To enable the verification of requirement properties, once the mode chart is constructed, it is converted into a NuSMV input model. For this high level NuSMV model, four modules are constructed to represent the system main module and each functional module (ABSBraking, Electrical, and Hydraulic). Among the requirement properties, safety requirements are often of primary concern. The process here aims to verify that the design goals are achieved, while ensuring that the model conforms to the safety requirements. Possible safety requirements that can be verified in this example are: "Driving assistance function(s) shall never hazardously interfere with the system state", "The system shall be able to withstand the occurrence of n failures, without entering a hazardous state", and "Dormant functions shall only be activated when needed". These requirements first have to be interpreted in terms of the behaviour specified in the mode chart model. For example, translating the first safety properties into: "The presence of the ABSBraking function shall not lead the system into Fail mode" and "The absence of the ABSBraking function shall not lead the system into Fail mode". These can be respectively expressed in CTL as:

```
! (AG (absB.Output = 1 -> SystemMode = BBW_Fail));
! (AG (absB.Output = 0 -> SystemMode = BBW_Fail));
```

The safety properties that can be verified are refined according to the refinement of the state machines. This refinement captures and retains the hierarchical composition of the model and allows more detailed verification to be performed. By examining the relationships between the dynamic behaviour of modules, it is now possible to verify more safety related requirements, from more straight-forward ones like "As long as Braking Energy ACT A is functioning, the Braking Energy function shall be present", or for a cold-standby system which examines the electrical and hydraulic modules: "Only either Electrical pressure or Hydraulic pressure shall be supplied at one time", to the effects of this function behaviour on the system modes, for example: " System shall not be allowed to enter hazardous mode when Electrical system is functioning".

4 Conclusions

There is an increasing need for early safety analysis that can guide system design, particularly in complex safety-critical systems. In this chapter, we have presented IACoB, a systematic method that integrates application of state-of-the-art CSA and BSA techniques from early stages. The method utilizes the synergies between the two techniques, and assists analysis of topological and behavioural models, verification of safety requirements, identification of design weakness, and systematic design of

degraded modes and fault tolerant strategies. An example of a brake-by-wire system was used to illustrate how the approach methodically achieves design improvements. In the context of this example, we demonstrated that it is possible to exploit the complementary strengths of CSA and BSA and achieve a combined application where the output of CSA creates useful input for BSA. The degree of automation enabled by the underpinning techniques allows analysis to be iterated, and contributes to a more-rigorous safety assessment. Future work includes application of the proposed concept in context of design using emerging architecture description languages such as AADL [6] and EAST-ADL [5].

Acknowledgements. This study was partly funded by the FP7 project MAENAD (Grant 260057).

References

[1] Adachi, M., Papadopoulos, Y., Sharvia, S., Parker, D., Tohdo, T.: An approach to optimization of fault tolerant architectures using HiP-HOPS. In: Software Practice and Experience. Wiley Interscience, Hoboken (2011), doi:10.1002/spe.1044; available online in advance of publication

[2] Arnold, A., Gerald, P., Griffault, A., Rauzy, A.: The Altarica formalism for describing concurrent systems. Fundamenta Informaticae 34, 109–124 (2000)

[3] Bozzano, M., Villafiorita, A.: The FSAP/NuSMV safety analysis platform. International Journal on Software Tools for Technology Transfer 9, 5–2 (2006), doi:10.1007/s10009-006-0001-2

[4] Colombo, D.: Brake-by-wire system development: technology and development process. Vehicle Systems-Active Control Systems Fiat Group (2007), http://staff.polito.it/enrico.canuto/Home_page/pdf/Incontro18gen2008/DColombo.pdf (accessed February 16, 2011)

[5] Cuenot, P., Chen, D., Gérard, S., Lönn, H., Reiser, M.-O., Servat, D., Kolagari, R.T., Törngren, M., Weber, M.: Towards improving dependability of automotive systems by using the EAST-ADL architecture description language. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems IV. LNCS, vol. 4615, pp. 39–65. Springer, Heidelberg (2007)

[6] Feiler, P.H., Gluch, D.P., Hudak, J.J.: The architecture analysis and design language (AADL): an introduction (2006), http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA455842 (accessed February 16, 2011)

[7] Grunske, L., Kaiser, B., Papadopoulos, Y.: Model-driven safety evaluation with state-event-based component failure annotations. In: Heineman, G.T., Crnković, I., Schmidt, H.W., Stafford, J.A., Ren, X.-M., Wallnau, K. (eds.) CBSE 2005. LNCS, vol. 3489, pp. 33–48. Springer, Heidelberg (2005)

[8] Hedenetz, B., Belschner, R.: Brake-by-wire without mechanical backup using a TTP-Communication network. Society of automotive engineering, SAE 981109 (1998), http://www.vmars.tuwien.ac.at/projects/xbywire/projects/new-BBW.html (accessed February 16, 2011)

[9] Heimdahl, M.P.E.: Formal model-based development in aerospace systems: challenges to adoption. Lecture notes, Software Engineering Center Critical Systems Research Group. University of Minnesota (2007)
[10] Joshi, A., Whalen, M., Heimdahl, M.P.E.: Model-based safety analysis final report. University of Minnesota Advanced Technology Centre (2006)
[11] Kaiser, B., Liggesmeyer, P., Mackel, O.: A new component concept for fault trees. In: Proceedings of the 8th Australian Workshop on Safety Crucial Systems and Software, vol. 33, pp. 37–46 (2003)
[12] Papadopoulos, Y.: Safety analysis of a distributed brake-by-wire systems for cars. ESPRIT 23396 (TTA) Deliverable, University of York (1998)
[13] Walker, M., et al.: Review of relevant safety analysis techniques. Traffic Efficiency and Safety through Software Technology Phase 2 ATESST2 Report, University of Hull (2008)
[14] Walker, M., Papadopoulos, Y.: PANDORA: The time of priority AND gates. In: IFAC Symposia on Information Control Problems in Manufacturing, pp. 237–242 (2006)

Reliability Analysis of Electronic Protection Systems Using Optical Links

Mirosław Siergiejczyk[1] and Adam Rosiński[2]

[1] Warsaw University of Technology, Faculty of Transport, Department Telecommunication in Transport, ul. Koszykowa 75, 00-662 Warsaw, Poland
e-mail: msi@it.pw.edu.pl
[2] Warsaw University of Technology, Faculty of Transport, Department Telecommunication in Transport, ul. Koszykowa 75, 00-662 Warsaw, Poland
e-mail: adro@it.pw.edu.pl

Abstract. Theory of the systems reliability is particularly applicable to electronic protection systems (alarm systems), which due to their specific character of use, should be characterised by the high level of reliability. The devices and electronic units applied in the wide range in those systems, the microprocessor systems in particular, require a new perspective on the reliability and the safety of the systems. The paper presents a reliability analysis of the electronic protection systems using optical links.

1 Introduction

Electronic protection systems realize the service safety assurance while travelling, which is one of the services that are realized by telematics transport systems [14,15]. This service can be realized by the systems installed at: airport, railway stations, logistic centres, trans-shipping terminals as well as by the systems installed in the mobile objects (e.g. vehicles). Suitability assurance is the essential condition of their correct operation.

The group of electronic protection system includes as follows:

- Intruder alarm system,
- Access Control System,
- Closed Circuit TeleVision,
- Fire Alarm System,
- External Terrains Protection System.

Protection resulting from operation of the systems can be provided by the following features:

- signalisation of health condition and personal danger,
- signalisation of environmental dangers,
- against-theft,
- vehicles location systems.

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 193–203.
springerlink.com

The intruder alarm system will be introduced in the following part of my paper, but similar issues can also be found in other electronic safety systems.

The European Standard EN 50131-1:2006 "Alarm Systems – Intrusion and Hold-up Systems – Part 1: System Requirements", which has also the status of the Polish Standard PN-EN 50131-1:2009 "Alarm Systems – Intrusion and Hold-up Systems – System Requirements" contains a list of definitions and abbreviations that are then used in subsequent chapters of this standard [5]. Among them there are definitions, such as:

- alarm system – electric installation, responsible for manual or automatic detection of the presence of danger,
- control and indicating equipment – a device for data receiving, processing, controlling, imaging, and further transmission thereof.

Alarm control panels are specialised devices that are meant to:

- receive information signals (analogue and/or digital) from various devices,
- process in accordance with a pre-programmed settings (of the installer and/or the manufacturer)
- control by specifying the appropriate output signals,
- provide imaging of events that occur on the respective devices of the anti-burglary system,
- transmit data to other systems (such as e.g. Alarm Receiving Centre, abbreviated ARC).

PN-EN 50131-1:2009 „Alarm Systems – Intrusion and Hold-up Systems – Part 1: System Requirements" defines the class of protection that the intruder alarm systems should meet. They are as follows:

- grade 1: low risk (it is assumed that the intruder has minimum knowledge about the alarm system and possesses easily accessible tools of the limited choice),
- grade 2: low-to-medium risk (it is assumed that the intruder has a minimum knowledge of the alarm system and has a widely available tools and portable devices such as digital multimeter),
- grade 3: medium-to-high risk (it is assumed that the intruder knows the alarm system entirely and has a complex set of powerful tools and portable electronic equipment),
- grade 4: high risk (applicable whenever safety has priority over all other factors. It is assumed that the intruder has the ability or resources to plan a burglary in detail and has a set of any equipment, including measures to replace the key of an electronic alarm system).

Having specified what class of the protection the intruder alarm system has to fulfil, there are selected devices that meet those requirements. The standard obviously refers to what units have to be applied. Therefore, there are various solution designs of the alarm control panel. They can fulfil the requirements of a specific class of protection, but they also differ among themselves depending on their manufacturer.

As it has already been mentioned, the alarm control panel is the „heart" of the intruder alarm system. Data is sent about the condition of individual supervisory lines (e.g. detectors), exit lines (e.g. load outputs) or a specific one introduced by the user or a maintenance guy (and earlier during the installation of the system). Information can directly be sent to the plate of the main alarm control panel, depending on the type of alarm control panel or also to modules, realising definite functions (e.g. expanding input, expanding output, interfaces of printers, etc.). Information between alarm control panel and individual modules is sent digitally using the transmission format that is mostly applied at present RS-232 or RS-485 or another one (very often elaborated by the manufacturer) [6,9]. There are also solutions of the burglary-signalling systems where transmission bus can combine:

- intra-several alarm control panels (they operate in the so-called annulus),
- control (e.g. the keyboard steering),
- alarm control panels with the supervisory and management centre as well as the managing of the integrated safety system.

The intruder alarm system can be divided into three principal groups:

- concentrated systems,
- distracted systems,
- mixed systems

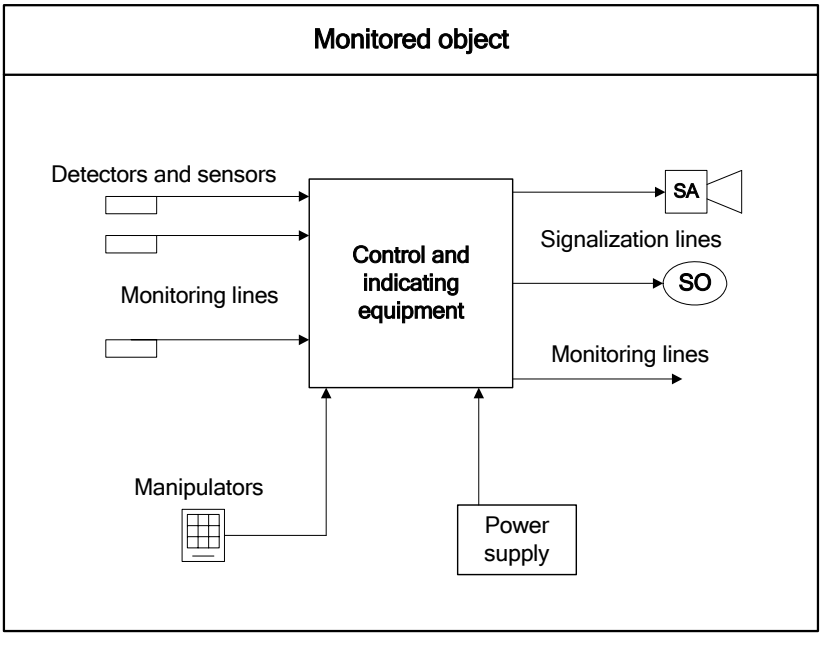


Fig. 1 Intruder alarm system with concentrated structure, where: SA – audio signalling device, SO – optical signalling device

The concentrated structures envisage connection of all the monitored lines and output lines (monitoring and signalising) to the alarm exchange (Fig. 1.).

In the widespread objects requiring a big number of monitoring lines and a big number of control zones, the systems basing on the microprocessor digital exchanges with the concentrated structure are not applicable. Therefore, there systems with dissipated or mixed structure must be used. A characteristics for the dissipated structure is decentralisation of the alarm exchange, basing on the use of transmission buses that are connected to the respective modules (input, output, power) as well as the use of transmission buses to connect the separate concentrated exchanges among themselves and thus creating the system with a dissipated structure. The mixed structure combines characteristics of both described here structures, and it means that the monitoring lines are connected both to the alarm exchange and to the expanding modules.

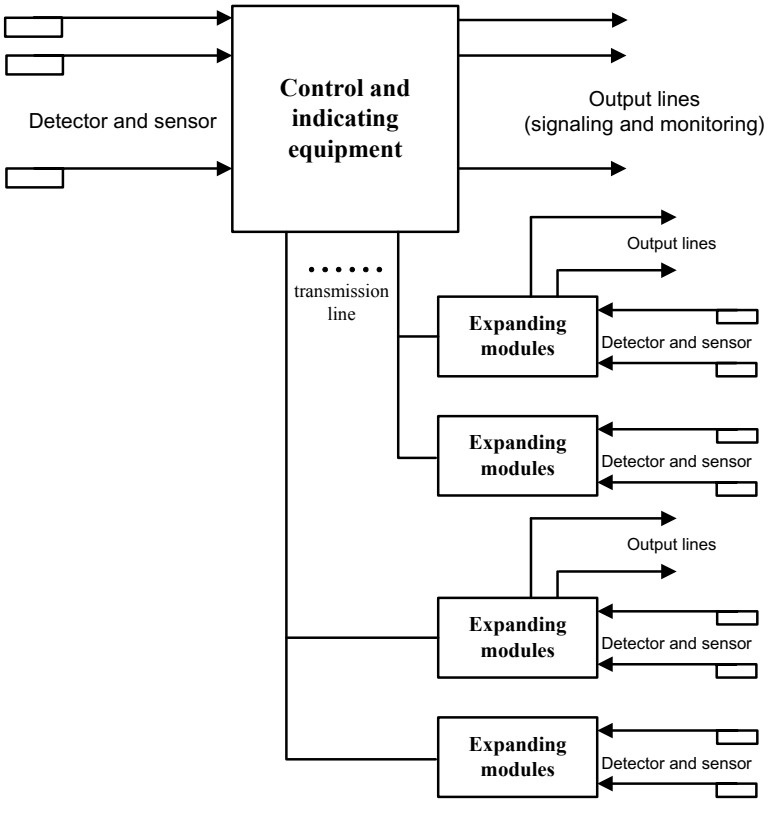


Fig. 2 Intruder alarm system – mixed systems

Figure 2 presents the mixed alarm system with the distracted character where own (switch boards) entries of the supervisory lines are used. The system where concentrated systems are connected by the RS-232 or RS-485 lines (or others, but

enabling data transmission between respective alarms and controls in the concentrated version, thus creating the intruder alarm system in the mixed version) can also be treated as the mixed alarm system.

The questions of reliability, exploitation and electromagnetic compatibility in the electronic safety systems are particularly essential, especially if they are they applied in domain of transport. There is very limited number of publication which present this issue [3,4,10]. However they do not take into account the reliability analysis of electronic safety systems in which the optical transmission was applied. That is why it seems necessary to consider such solutions as well.

2 Analysis of Electronic Protection Systems Using Optical Links

Electronic protection system has a defined reliability structure: serial, mixed or parallel. In general, it is presented in Fig. 3. Such a structure is often applicable in large and extensive objects. The reliability analysis of this type of structures is presented in many scientific papers [1,8,11,12,16,17].

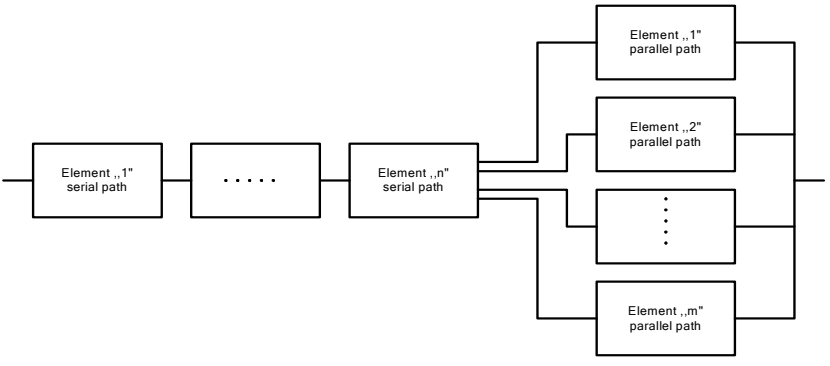


Fig. 3 Structural reliability flow chart of electronic protection system

Due to a specific characteristics of the protected objects (e.g. airports, railway stations, logistic bases), as those buildings are very often located on a large area and simultaneously they have a huge enough surface, there is a need to design the Intruder Alarm Systems which shall enable placement of component units in the protected rooms and adjacent terrains. Using the conventional line solutions in which transmission lines are applied (e.g. modules, manipulators) to the transmission of electric signals is not sufficient because of the guaranteed quality of the data transmission in the function of distance among the units of the system. Electromagnetic disturbances that may occur are the next essential issue. That is why the transmission measure, namely the optical fibre, started to be applicable.

Data transmission requires conversion of electrical signals into optical (fibre-optic transmitter) and vice versa (fibre-optic receiver) [6] - Figure 4. Since data

transfer information in the electronic security system busses is normally bi-directional, so the fibre optic converter system should include both the transmitting and the receiving system. Therefore, two optical fibres are necessary to ensure data transmission between the two converters.

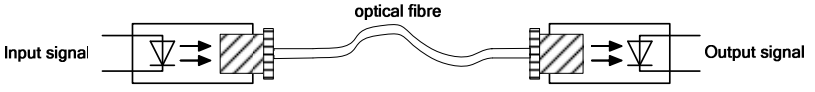


Fig. 4 Optical fibre link for signal transmission

The advantages of fibre optic transmission between devices forming the Intruder Alarm System include inter alia [13]:

- high resistance of communication to interferences,
- no generation of electromagnetic interference,
- lack of sensitivity to the phenomenon of stray currents (this is particularly important in the railway environment where in close proximity to each other there may be a small capacities of say milliwatts (telecommunication signals) and large capacities of say megawatts (electric locomotives),
- high bandwidth fibre enables the connection of further devices,
- galvanic isolation of devices.

By modelling the single optical bus transmission lines applying the serial structure, the readiness should be considered of such component units as: amplifiers, cables, regenerators, etc. (Fig. 5) [7].

Let us assume the following indications of the value of the readiness coefficients: transponder K_{gtrans}, regenerator K_{greg}, amplifier K_{gwzm}. Analysing the process of the exploitation of the optical link, we can distinguish the following states of efficiency:

- s_0 – the state of the correct execution of the function of transmission,
- s_1 – the state in which the functions of the broadcast realisation are not executable.

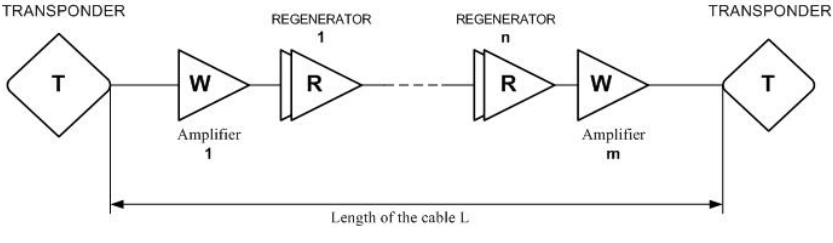


Fig. 5 Structure of a single optical transmission line

The matrix of transitions probabilities between the distinguished states takes the form of:

$$\mathbf{P} = \begin{bmatrix} 1-\lambda_k & \lambda_k \\ \mu_k & 1-\mu_k \end{bmatrix} \tag{1}$$

Accepting the solid intensity of damages λ for individual units of the optical link and the solid intensity of the service μ, we can determine a stationary value of the readiness coefficient of optical amplifier, regenerator, and transponder in the form of:

$$K_{g\,trans} = K_{g\,reg} = K_{g\,wzm} = P_0 = \frac{\mu_k}{\mu_k + \lambda_k} \tag{2}$$

where index *k* means parameters of time distribution of proper operation and repair time respectively for optical amplifier, regenerator, and transponder, respectively.

Components such as fibre optic cables also have a significant impact at operational readiness of the entire optical link. The optical cables readiness can be counted using the *CC Cable Cut* parameter, which expresses the average length of the cable that breaks once during the whole year (8760 [h]). Coefficient *CC is expressed* in kilometres, meanwhile the value of the parameter $MTBF_K$ *(Mean Time Between Failure)* for the cable whose length is *L*, is defined in hours and has a form of [2]:

$$MTBF_K(L) = \frac{CC \cdot 8760}{L} \tag{3}$$

The value of the readiness coefficient for the optical cable can be written in the form of:

$$K_{gK} = P_0 = \frac{MTBF_K}{MTBF_K + MTTR_K} \tag{4}$$

where: $MTTR_K$ - is the optical cable repair time.

3 Analysis of Practical Application Reliability of Electronic Protection Systems Using Optical Links

Figure 6 shows a diagram of the Intruder Alarm System with mixed structure, which has been designed and implemented using a microprocessor alarm control panel INTEGRA.

The hereto presented system belongs to the group of mixed systems, i.e. part of monitoring lines (e.g. PIR detector, magnetic sensor, alarm box) is connected by radio channels with a special module of wireless devices. The module is connected to the mainboard of the alarm control panel via the wired transmission bus. Also, some

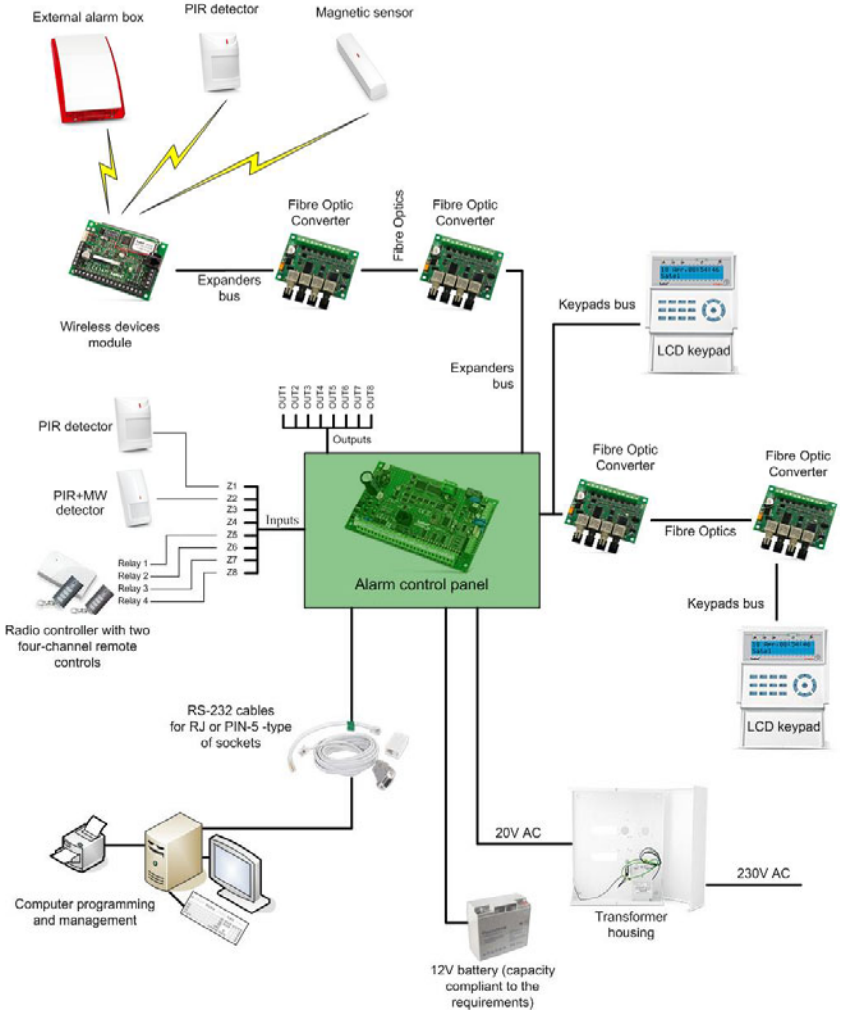


Fig. 6 The intruder alarm system of dispersed structure (with applied fibre optic converters)

of the detectors are linked to the mainboard via a conventional monitoring wired-lines. The entire system is programmable and controllable by a computer (using appropriate software) linked to the mainboard of the alarm control panel via RS-232 interface. The system is also operable through LCD keypads. One of them is directly connected to the mainboard of the alarm control panel via conventional wired bus keypads. The second one is also connected to the keypads bus, but using the fibre optic converters between which data transmission takes place through the transmission medium, namely the fibre-optic cable. There are neither amplifiers nor regenerators used in the hereto applied solution.

The following values have been adopted in the analysed system:

- research time – 1 year:

$$t_b = 8760[h]$$

- reliability of fibre-optic converter:

$$R_{ks}(t_b) = 0{,}99$$

- intensity of repairs of fibre-optic converter (it corresponds to the repair time equal to 12 [h]):

$$\mu_{ks} = 0{,}08333\left[\frac{1}{h}\right]$$

- repair time of fiber-optic cable:

$$MTTR_K = 24[h]$$

- fiber-optic cable intersection parameter:

$$CC = 4[km]$$

- length of fibre-optic cable:

$$L = 2[km]$$

Knowing the value of reliability $R_{ks}(t_b)$, we may estimate the intensity of fibre-optic converter damages λ_{ks}. The following relationship can be used for the exponential distribution:

$$R_{ks}(t_B) = e^{-\lambda_{ks} t_B} \text{ for } t \geq 0$$

therefore:

$$\lambda_{ks} = -\frac{\ln R_{ks}(t_B)}{t_B}$$

For $t_b = 8760[h]$ and $R_{ks}(t_b) = 0{,}99$ we receive:

$$\lambda_{ks} = -\frac{\ln R_{ks}(t_B)}{t_B} = -\frac{\ln 0{,}99}{8760} = \frac{0{,}01}{8760} = 1{,}147298 \cdot 10^{-6}\left[\frac{1}{h}\right]$$

Knowing the value of λ_{ks}, the expected operation time between successive damages is calculable:

$$E(T) = \frac{1}{\lambda_{ks}} = 871612[h]$$

Fibre-optic converter readiness index can be determined from the following dependence (2):

$$K_{g\,ks} = \frac{\mu_{ks}}{\mu_{ks} + \lambda_{ks}} = \frac{0{,}08333}{0{,}08333 + 1{,}147298 \cdot 10^{-6}} = 0{,}999986$$

The readiness index of the fibre-optic link can be determined from the dependence (3 and 4):

$$MTBF_K(L) = \frac{CC \cdot 8760}{L} = \frac{4 \cdot 8760}{2} = 17520[h]$$

$$K_{gK} = \frac{MTBF_K}{MTBF_K + MTTR_K} = \frac{17520}{17520 + 24} = 0{,}998632$$

The readiness index of the entire single fibre-optic link is:

$$K_g = K_{gks} \cdot K_{gK} \cdot K_{gks} = 0{,}999986 \cdot 0{,}998632 \cdot 0{,}999986 = 0{,}998604$$

4 Conclusions

Not only the stage of the threat effecting from an object, designed according to the currently binding standards and recipes, but also a possibility to use modern solutions in the area of safety engineering should be considered when designing an electronic safety protection system (it has been presented in the Report on Exemplary Intruder Alarm System). The example is a possibility to utilize optical units as elements assuring data transmission between the alarm control panel and the modules. This increases the level of the guaranteed quality of data transmission in the function of distance between elements of the system, as also it protects the transmitted information against electromagnetic disturbances that may occur.

The paper presents methodology for analysing reliability of those electronic protection systems where optical links have been applied. This type of the consideration are particularly important in the event when this type of technical solutions are applied to the protection of objects about the strategic meaning for the country (e.g. airports, railway stations, atomic power stations) and its defence (e.g. military base). The results obtained from the reliability analysis can be used while designing of the system in order to assure the suitable values of the reliability coefficients. There is also a possibility to use the methodology hereto presented to analyse the already existing systems in order to qualify the influence that modernisation of the system units has on their reliability.

References

[1] Będkowski, L., Dąbrowski, T.: The basis of exploitation, part II: The basis of exploational reliability. Wojskowa Akademia Techniczna, Warsaw (2006)

[2] Chołda, P., Jajszczyk, A.: Assessment of availability in telecommunication networks. Telecommunication Review and Telecommunication News. No. 2-3/2003. Publication by Sigma NOT, Warsaw (2003)

[3] Dyduch, J., Paś, J.: Exploitation of the transport systems of supervision on the extensive railway area. In: VII The National Conference: the Technical Diagnostics of Devices and Systems – DIAG 2009, Ustroń (2009)

[4] Dyduch, J., Paś, J.: Electromagnetic environment on the railway and its influence on the systems of the safety. Transport and Communication (1)2009)

[5] European Standard EN 50131-1:2006. Alarm Systems – Intrusion and Hold-up Systems – Part 1: System Requirements. Brussels: European Committee for Electrotechnical Standardization CENELEC

[6] Haykin, S.: Telecommunication systems, vol. I & II. WKiŁ, Warsaw (2004)

[7] Horowitz, P., Hill, W.: The art of electronics, vol. I & II. WKiŁ, Warsaw (2006)

[8] Jaźwiński, J., Ważyńska-Fiok, K.: System safety. PWN, Warsaw (1993)

[9] Norman, T.: Integrated security systems design. Butterworth Heinemann, Butterworths (2007)

[10] Paś, J., Dyduch, J.: Influence of the electromagnetic disturbances on the transport systems of safety. Measurements Automation Robotics (9,10) (2009)

[11] Rosiński, A.: Reliability analysis of the electronic protection systems with mixed – three branches reliability structure. In: Proc. International Conference European Safety and Reliability (ESREL 2009), Prague, Czech Republic, pp. 1637–1641 (2009)

[12] Rosiński, A.: Design of the electronic protection systems with utilization of the method of analysis of reliability structures. In: Proc. Nineteenth International Conference On Systems Engineering (ICSEng 2008), Las Vegas, USA, pp. 421–426 (2008)

[13] Siergiejczyk, M., Gago, S.: A Concept of Monitoring and Supervising System in Railway Junction. In: Sixth International Scientific & Technical Conference LOGITRANS 2009, Szczyrk (2009)

[14] Siergiejczyk, M.: Maintenance Effectiveness of Transport Telematics Systems. Transport Series, vol. (67). Scientific Works of the Warsaw University of Technology, Warsaw (2009)

[15] Wawrzyński, W., Siergiejczyk, M., et al.: Final Report on Grant KBN 5T12C 066 25. Methods for Using Telematic Measures to Support Realisation of Transport Tasks, Supervisor: Associate Professor Ph.D. D.Sc. W. Wawrzyński, Warsaw (2007)

[16] Ważyńska-Fiok, K., Jaźwiński, J.: Reliability of technical systems. PWN, Warsaw (1990)

[17] Zamojski, W. (ed.): Reliability and Maintenance of Systems. Publisher of Wroclaw University of Technology, Wroclaw (1981)

Avoiding Probability Saturation during Adjustment of Markov Models of Ageing Equipment

Jarosław Sugier

Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland
e-mail: jaroslaw.sugier@pwr.wroc.pl

Abstract. Markov models are well established technique used widely for modeling equipment deterioration. This work presents an approach where Markov models represent equipment ageing and also incorporate various maintenance activities. Having available some basic model it is possible to adjust its parameters so that it represents some hypothetical new maintenance policy and then to examine impact that this new policy has on various reliability characteristics of the system. The paper deals with a method of model adjustment and specifically investigates its one particular problem: avoiding probability saturation in a model which is tuned towards increased repair frequencies. The text describes the adjustment method in a general case, identifies specific risk of probability saturation that may take place during the iterative procedure and proposes a new extension to the method that overcomes this problem with minimal intervention in the internal structure of the model, in a specific class of cases.

1 Introduction

Selection of an efficient maintenance strategy plays a very important role in the management of today's complex systems. When searching for an optimal strategy, numerous issues must be taken into account and, among them, reliability and economic factors are often equally important. Finding a reasonable balance between them is the key point in efficient maintenance management and to facilitate finding such a balance some measures should be available that allow quantitative evaluation of the deterioration process of a system in a case when it is subjected to various maintenance actions (inspections, repairs, replacements, etc.).

This work deals with development that aims at providing a computer tool for a person deciding about maintenance activities, which would help in evaluation of both the risks and the costs associated with selection of various possible maintenance strategies. Rather than searching for a solution to a problem: "what maintenance strategy would lead to the best reliability and dependability parameters of the system operation", in this approach different maintenance scenarios can be examined in the "what-if" type of studies and then, using the tool, their reliability

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 205–217.
springerlink.com

and economic effects can be automatically estimated so that the person responsible for the maintenance is assisted in making an informed decision ([5], [7], [11], [20]).

The proposed application of Markov models in representation of deterioration and maintenance processes has been presented initially in [4], while in [14] and [15] the procedure of model adjustment to modified repair frequencies was discussed. Efficiency of this method with its possibly weak points was further investigated in [17] and [18]. In this work, we extend the method so that it can properly deal with a class of cases when so called probability saturation takes place during adjustment towards increased frequencies of repairs.

2 Adjusting the Deterioration Model

There are three major factors that decide about equipment deterioration: its physical characteristics, operating practices, and the maintenance policy. Of these three aspects, especially the last one relates to the events and actions that should be properly modeled.

2.1 Construction of the Model

The method discussed in this work is based on the model in which the equipment will deteriorate in time and, if not maintained, will eventually fail. If the deterioration process is discovered, preventive maintenance is performed which can restore the condition of the equipment ([1], [10]). Such a maintenance activity will return the system to a specific state of deterioration, whereas repair after failure will restore to "as new" condition. The maintenance components that must be recognized in the model are: monitoring or inspection (how the equipment state is determined), the decision process (which determines the outcome of the decision), and finally, the maintenance actions (or possible decision outcomes). These elements can be properly incorporated in an suitable state-space (Markov) model ([6], [8], [9], [12], [13], [19]) which consists of the states the equipment can assume in the process, and the possible transitions between them. In a Markov model, the rates associated with the transitions are assumed to be constant in time.

The method described in this work uses specific model developed for the Asset Maintenance Planner (AMP) ([2], [3]). The AMP model is designed for equipment exposed to deterioration but undergoing maintenance at prescribed times. It computes the probabilities, frequencies and mean durations of the states of such equipment. The basic ideas in the AMP model are the probabilistic representation of the deterioration process through discrete stages, and the provision of a link between deterioration and maintenance. For structure of a typical AMP model see Fig. 1.

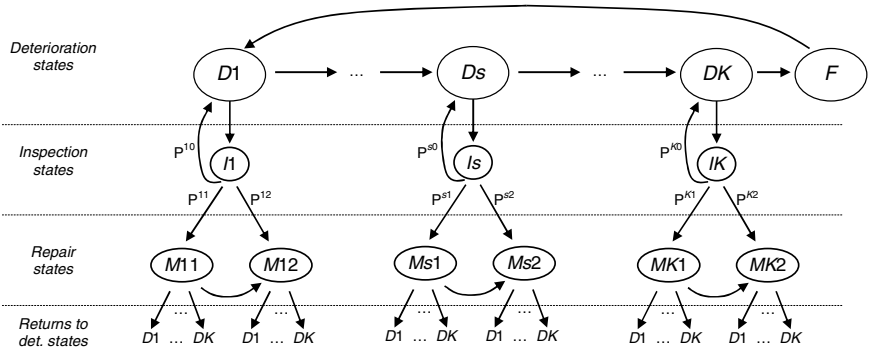


Fig. 1 The state-transition model representing the deterioration process with inspection and repair states (an example with two repair types

In this model, the deterioration progress is represented by a chain of *deterioration states* $D1 \ldots DK$ which leads to the *failure state* F. In most situations, it is sufficient to represent deterioration by three stages: an initial ($D1$), a minor ($D2$), and a major ($D3$) stage ($K = 3$). This last is followed, in due time, by equipment failure (F) which requires extensive repair or replacement.

In order to slow deterioration and thereby extend equipment lifetime, the operator will carry out maintenance according to some pre-defined policy. In the model of Fig. 1, regular inspections (Is) are performed which result in decisions to continue with minor ($Ms1$) or major ($Ms2$) maintenance or do nothing (more than two types of repairs can also be included). The expected result of all maintenance activities is a single-step improvement in the deterioration chain; however, allowances are made for cases where no improvement is achieved or even where some damage is done through human error in carrying out the maintenance, which results in returning to the stage of more advanced deterioration.

The choice probabilities (at transitions from inspection states) and the probabilities associated with the various possible outcomes are based on user input and can be estimated, e.g., from historical records or operator expertise.

Mathematically, the model expressed in Fig. 1 can be represented by a semi-Markov process, and solved by the well-known procedures. The solution will yield all the state probabilities, frequencies and mean durations. Another technique, employed for computing the so-called first passage times (FPT) between states, will provide the average times for first reaching any state from any other state. If the end-state is F, the FPTs are the mean remaining lifetimes from any of the initiating states.

2.2 Adjusting the Model to Requested Repair Frequencies

Preparing the Markov model for some specific equipment is not an easy task and requires expert intervention. The goal is to create the model representing closely the real-life deterioration process known from the records that usually describe

equipment operation under a regular maintenance policy with some specific frequencies of inspections and repairs. The model itself permits calculation of the repair frequencies and compliance of the computed and recorded frequencies is a very desirable feature that verifies trustworthiness of the model.

In this section, we will summarize the method of model adjustment proposed in [14] and [15] that aims at reaching such a compliance. It can be used also for a different task: fully automatic generation of a model for some new maintenance policy with modified frequencies of repairs. Such a task needs to be done fairly often during evaluation of various maintenance scenarios.

Let K represents the number of deterioration states and R the number of repairs in the model under consideration. Also, let P^{sr} = probability of selecting maintenance r in state s (assigned to the decision after state Is) and P^{s0} = probability of returning to state Ds from inspection Is (situation when no maintenance is scheduled as a result of the inspection). In Fig. 1 the probabilities are located nearby respective transitions. Then, for all states $s = 1 \ldots K$:

$$\mathrm{P}^{s0} + \sum_{r} \mathrm{P}^{sr} = 1 \tag{1}$$

Let F^{r} represent the frequency of repair r acquired through solving the model. The problem of model tuning can be formulated as follows:

> Given an initial Markov model M_0, constructed as above and producing the initial frequencies of repairs $\mathbf{F}_0 = \left[\mathrm{F}_0^1, \mathrm{F}_0^2, \ldots \mathrm{F}_0^R\right]$, adjust the probabilities P^{sr} so that some goal frequencies $\mathbf{F}_\mathrm{G}$ are achieved.

The vector $\mathbf{F}_\mathrm{G}$ usually represents the observed historical values of the frequencies of various repairs.

In the proposed solution, a sequence of tuned models $M_0, M_1, \ldots M_N$ is evaluated with each consecutive model approximating desired goal with a better accuracy. The procedure consists of the following steps repeated in an iterative loop with i denoting the iteration counter:

1° For the current model M_i, compute the vector of repair frequencies $\mathbf{F}_i$.

2° Evaluate an error of M_i as a distance between vectors $\mathbf{F}_\mathrm{G}$ and $\mathbf{F}_i$.

3° If the error is within the user-defined limit, consider M_i as the final model and stop the procedure ($N = i$); otherwise continue with the next step.

4° Construct a new model M_{i+1} through adjusting values of P_i^{sr} and compute P_{i+1}^{s0} from equation (1).

5° Proceed to step 1° with the next iteration.

The error computed in step 2° can be expressed in many ways. As the frequencies of repairs may vary in a broad range within one vector $\mathbf{F}_i$, yet the values of all are significant in model interpretation, the relative measures work best in practice. The most restrictive formula evaluates maximum relative error over all frequencies and this was used in this work:

$$\left\|\mathbf{F}_G - \mathbf{F}_i\right\| = \max_r \left|F_i^r / F_G^r - 1\right| \tag{2}$$

2.3 Tuning Repair Probabilities

Of all the steps outlined in the previous point, it is clear that adjusting probabilities P_i^{sr} in step 4° is the heart of the whole procedure.

In general, the probabilities represent $K \cdot R$ free parameters and their uncontrolled modification could lead to serious deformation of the model. To avoid this, a restrictive assumption is made: if the probability of some particular maintenance must be modified, it is modified proportionally in all deterioration states, so that at all times

$$P_0^{1r} : P_0^{2r} : \ldots : P_0^{Kr} \sim P_i^{1r} : P_i^{2r} : \ldots : P_i^{Kr} \tag{3}$$

for all repairs ($r = 1 \ldots R$).

This assumption also significantly reduces dimensionality of the problem, as now only R scaling factors $\mathbf{X}_{i+1}=[X_{i+1}^1, X_{i+1}^2, \ldots X_{i+1}^R]$ must be found to compute new probabilities for the model M_{i+1}:

$$P_{i+1}^{sr} = X_{i+1}^r \cdot P_0^{sr}, \quad r = 1 \ldots R, \; s = 1 \ldots K \tag{4}$$

Moreover, although the frequency of a repair r depends on the probabilities of all repairs (modifying probability of one repair changes, among others, state durations in the whole model; thus, it changes the frequency of all states), it can be assumed that, in a situation of a single-step small adjustment, its dependence on repairs other than r can be considered negligible and F_i^r can be considered to be a function of just one variable:

$$F_i^r = F_i^r\left(X_i^1, X_i^2 \ldots X_i^R\right) \approx F_i^r\left(X_i^r\right) \tag{5}$$

With these assumptions, generation of a new model is reduced to the problem of solving R non-linear equations in the form of $F_i^r\left(X_i^r\right) = F_G^r$ and this task can be accomplished with one of the standard root-finding algorithms.

One point of the procedure requires additional attention, though: applying equation (4) with $X_{i+1} > 1$ may violate condition

$$\sum_{r=1}^{R} P_{i+1}^{sr} \le 1 \tag{6}$$

in some deterioration state s. This situation needs special tests that would detect such illegal probability values and reduce them proportionally so that their sum does not exceed 1: a so called *scale-down transformation* needs to be applied. As practical studies show such conditions do occur during model tuning towards repair frequencies that are remarkably higher than F_0^r from the initial model M_0. In its simplest form, the scale-down operation consists in dividing each probability P^{sr} in the offending state s by the sum of all repair probabilities in this state:

$$\mathrm{P}^{sr} = \mathrm{P}^{sr} \big/ S_{Ds}, \quad S_{Ds} = \sum_{r=1}^{R} \mathrm{P}^{sr} \tag{7}$$

This will also lead to $\mathrm{P}^{s0} = 0$ which means that every inspection ends with some repair and there are no direct returns from *Is* state to *Ds*. Moreover, this obligatory correction mechanism can result in violation of the proportionality rule (3) as an inevitable side effect.

The following three approximation algorithms were implemented in the task of solving equation (5): Newton method working on a linear approximation of $\mathrm{F}_i^r()$ functions (the NOLA method), the secant method and the false position (*falsi*) method. For their detailed presentation please refer to [14] and [15].

Generally, the practical tests have shown that although simplifications of the NOLA solution may seem critical, it is reasonably efficient and stable in real-world practical cases because it has one advantage over its more sophisticated rivals: since it does not depend on previous approximations, selection of the starting point is not so important and the accuracy during the first iterations is often better than in the secant or *falsi* methods. Superiority of the latter methods, especially of the *falsi* algorithm, manifests itself in the later stages of the approximation when the potential problems with an initial selection of the starting points have been diminished.

3 Automatic Correction of the Model in the Case of Probability Saturation

As practical applications of the adjustment method described in the previous section have shown, the procedure must be carefully applied to the models that represent real-life deteriorating processes because it is relatively easy to arrive at the solution that correctly realizes the optimization goal, i.e. produces the requested repair frequencies, but the internal structure of the model is modified to the degree which harms the relation between the new unit and the original equipment. In this section we will propose an approach that aims at one specific problem related to this issue which may arise in practical cases when increasing the repair frequencies is requested.

Adjusting the model to the repair frequencies that are substantially higher than the original ones may lead to *model saturation* – a condition in which repair probabilities reach the limit (6) and there is no room for further increase if the adjustment procedure is limited only to the simple probability scaling expressed by equation (4). In this situation bringing together the two requirements: tuning the model towards high repair frequencies and, at the same time, keeping the modifications of the internal structure within a safe range that does not break proper relation with the original, is a challenge that needs a new, careful development.

3.1 Application Context

The method of model adjustment that is being considered in this work has been practically implemented in the Asset Risk Manager (ARM) software system which

uses the concept of a life curve and discounted cost to study the effect of equipment ageing under different hypothetical maintenance strategies ([4], [18]). As noted in section 2, the method uses semi-Markov models of the Asset Maintenance Planner (AMP) ([2], [3]). For the ARM program to automatically generate the life curves for different requested maintenance policies (with, among other parameters, different repairs frequencies), default Markov model for the equipment has to be built and stored in the computer database. This is done through the prior running of the AMP program by an *expert user*. Therefore, both AMP and ARM programs are closely related, and usually, should be run consecutively.

Implementation details of Markov models, tuning their parameters and all other internal particulars should not be visible to the non-expert *end user* who actually operates the ARM software in order to investigate various potential modifications of the present (default) maintenance policies associated with the model and evaluates their economic and reliability costs. All final results are visualized either through an easy to comprehend idea of a life curve or through other well-known concepts of financial analysis. Still, prior to running the analysis some expert involvement is needed, largely in preparation, importing and adjusting AMP models. After that the adjustment method should run automatically in the background and the end user should be presented with results that come from the tuned models. In this context it is vital that the method can generate correctly adjusted models reliably and without human intervention.

Discussion included in [17] and [18] investigated main challenges that are brought by this task. It has shown that, while tuning the model towards decreased repair frequencies usually succeeds without additional specific requirements, some special rules in model construction should be respected if the model is to be tuned towards *increased* frequencies. The two main factors that were recognized were as follows: (1) although it may seem that in the initial (minor) deterioration state no repairs are performed after inspections, still some non-zero probabilities are required in $D1$ if purely hypothetical questions like "What if I start some repair twice as often as previously?" shall be allowed; (2) including an option of not doing any repair after inspection in the later deterioration states, albeit with small probability, is also desirable because it increases ability of the model to represent diverse maintenance configurations found in the studies.

3.2 The Problem of Model Saturation

For practical illustration two real-world Markov models were selected that are especially prone to the problems of probability saturation. Specifically, they do not follow rule (1) from the previous point: they assume that no repair is performed after inspections in the first deterioration state, i.e. $P^{1r} = 0$ and $P^{10} = 1$. Such assumption is common for AMP models created according to actual historical records describing equipment operation.

Both models have the same general structure with $K = R = 3$, i.e. they include three deterioration states ($D1 \div D3$) and three repairs: minor (index = 1), medium (2) and major (3). The main difference between them lies in distributions of repair

probabilities P^{sr} in the deterioration states (or, strictly speaking, in inspection states $I1 \div I3$ associated with the deterioration states; see Fig. 1). These probabilities are given in Table 1.

Table 1 Repair probabilities in Markov models used as examples A and B

Deterioration state:	$D1$			$D2$			$D3$		
Probability of repair:	P^{11}	P^{12}	P^{13}	P^{21}	P^{22}	P^{23}	P^{31}	P^{32}	P^{33}
Model A:	0.000	0.000	0.000	0.80	0.15	0.05	0.20	0.50	0.30
Model B:	0.000	0.000	0.000	0.64	0.12	0.04	0.18	0.45	0.27

The model A has been created with assumption that although there are no repairs in the first state $D1$, when the equipment is in subsequent states $D2$ and $D3$ every inspection leads to some sort of repair and in these states the totals $S_{D2} = S_{D3} = 1$ ($P^{20} = P^{30} = 0$). Looking at the probability distribution in each state it can be seen that in the medium deterioration $D2$ the minor repair is evidently the most often chosen one ($P^{21} = 0.80$) while in the major deterioration $D3$ the distribution is to some extent more balanced with medium repair taking half of the chances ($P^{32} = 0.50$).

The model B is a sibling of A with just one but important difference: repair probabilities in $D2$ and $D3$ are lower by, respectively, 20% and 10% than the values of model A, which also means that after inspections in these states it is possible to return to Ds without undertaking any repair ($P^{20} = 0.2$ and $P^{30} = 0.1$). In other words, model B, as opposite to model A, has been created according to the requirement (2) introduced in the previous point.

In the coming analyses series of models will be generated from the initial model M_0 in both cases A and B for a sequence of goal frequencies

$$\mathbf{F}_G = \left[\alpha \cdot \mathrm{F}_0^1, \mathrm{F}_0^2, \mathrm{F}_0^3\right] \tag{8}$$

with factor α increasing from 0.5 (frequency of minor repair reduced by half) to 2.0 (minor repair performed twice as often) in steps of 0.1. Values of α in the figures and in the following discussion will be expressed as %. Frequency of the minor repair (no. 1) was selected as the varying parameter of $\mathbf{F}_G$ just as an example with frequencies of other repairs remaining constant, but equivalent results could be demonstrated with changing frequencies of medium or major repairs. The figures will include graphs presenting variations of repair probabilities P^{sr} and their sums S_{Ds} in deterioration states of the final adjusted models as functions of the α factor.

The problem of probability saturation is illustrated in Fig. 2 which includes minor repair probabilities in all states (probability of other repairs are not included to preserve space) for models A and B tuned with the standard procedure described in the previous section. Both models can be successfully adjusted only up to the point of saturation which is reached for α = 100% for model A (i.e. the initial model is already saturated) and 130% for model B (as it turns out, in this particular case $P^{20} = 0.2$ and $P^{30} = 0.1$ leave space enough for 30% increase in frequency

of the minor repair). In both cases for these goals probabilities in states *D*2 and *D*3 sum up to unity and cannot be further increased, while in *D*1 the P^{11} is zero and applying the scaling factor as in equation (4) cannot produce any increase. On the other hand, the procedure has no problems with adjustment towards lower frequencies and in such cases the probabilities are scaled accordingly ([14], [15], [17]).

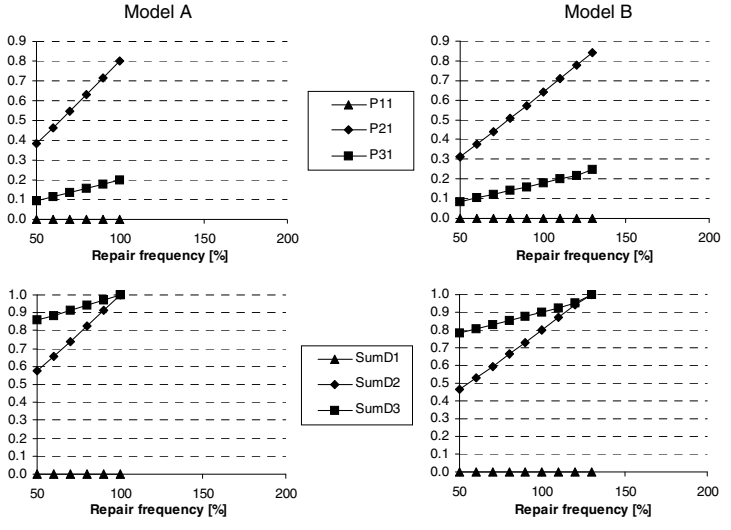


Fig. 2 Unsuccessful adjustment of the models with the standard (unmodified) procedure: probability of the minor repair in all three states (above) and sum of probabilities per state (below)

3.3 Challenges of Model Alteration

The above example of unsuccessful tuning can be used also for illustration of the main idea of the proposed extension to the algorithm: if the model gets saturated during the adjustment iteration but there is still some state with null repair probability, the process can be continued in the same iterative way after some non-zero probability is added into this state. Such modification, though, goes far beyond the restrictive assumption expressed by equation (3) and, being a more serious invasion into the model structure, must be applied in a cautious and thoughtful manner.

In particular, the following two issues must be taken into account: (1) forcing non-zero probability in some state before it is not absolutely necessary, i.e. prior to model saturation, instantly changes reaction to the adjustment iterations, hence may change the final result of the tuning also in cases when the standard procedure would be able to produce the correct result; (2) replacing the null value of P^{sr}, even if delayed up to the moment of saturation, but with probability which is too

high for the needs of the adjustment also may affect the final result in a way that is against the general idea of the conservative tuning which tries to preserve the structure of the original model with minimal possible modifications.

Figure 3 illustrates these two problems using model B as an example. The two upper graphs show adjustment results when null P^{sr} is replaced with a non-zero value right from the first iteration only if respective frequency needs to be increased in the goal vector, i.e. without waiting until the model gets saturated. In this case it means that the model is modified for cases where $\alpha > 100\%$ instead of $\alpha > 130\%$. As the graphs show, forcing $P^{11} > 0$ prematurely causes evident instabilities in growths of P^{21} and P^{31} and even more significant instabilities in distributions of sums S_{D1}, S_{D2} and S_{D3}. In fact the model does not reach saturation in state $D3$ even for $\alpha = 200\%$, while the Fig. 2 indicates that this model should saturate for $\alpha = 130\%$.

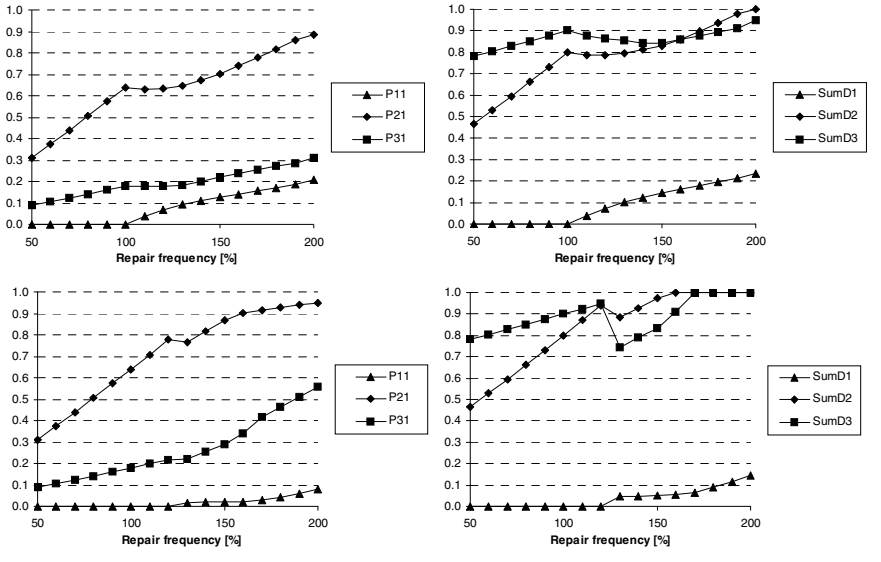


Fig. 3 Incorrect modifications generated by the adjustment procedure: non-zero repair probabilities introduced before model saturation (above) and too high probabilities forced after model saturation (below)

The two lower graphs in Fig. 3 present the results when the moment of probability increase is properly delayed until model saturation ($\alpha = 130\%$) but P^{11} is assigned with a value which exceeds the needs of tuning. Again, as a result the growths of P^{21} and P^{31} are noticeably disturbed and even more evident instabilities can be seen in graphs of the sums S_{Ds}: the exaggerated intervention applied for $\alpha = 130\%$ drives the model out of the saturation state until $\alpha = 170\%$, and only after this point the procedure continues with the expected linear growth of P^{11}.

3.4 Extension of the Adjustment Procedure

After analyses of case studies like the above two examples, the following modification of the adjustment procedure has been found to be the most flexible and efficient solution that gives optimal results in broad range of practical cases. It not only delays the increase of null probability until the moment of model saturation, but also scales its value adequately.

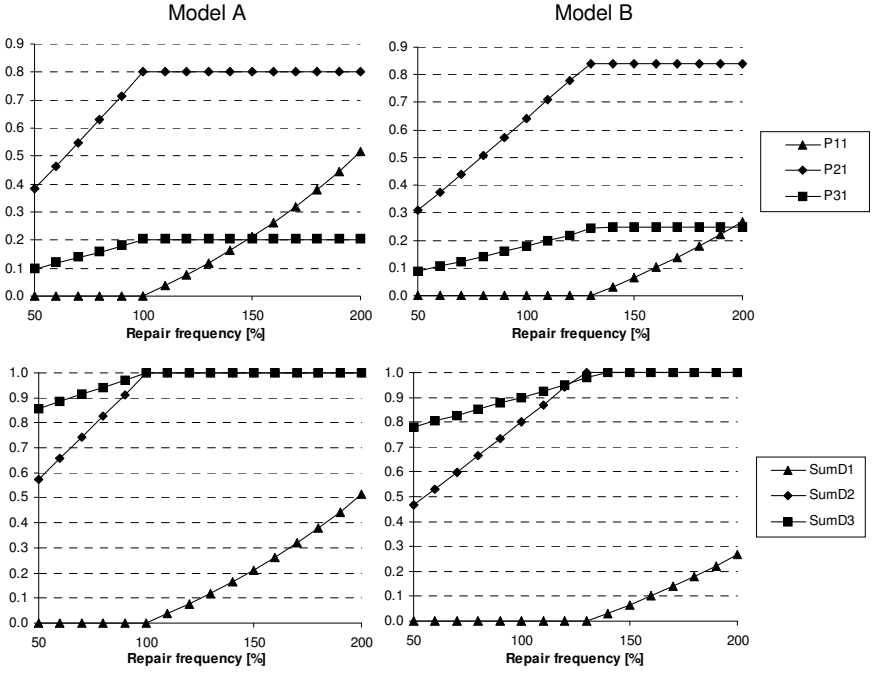


Fig. 4 Tuning the models A (left) and B (right) by the proposed extension of the adjustment procedure

The modification does not amend the general iterative scheme defined in point 2.2 (steps 1° ÷ 5°); the changes are limited only to internal details of step 4° which computes new probability values for the next model M_{i+1}. The modified implementation of this operation detects and deals in a different way with the following two cases:

(a) If the model is not saturated, i.e. there is a state with $0 < S_{Ds} < 1$, the standard approach is applied: in all states the values of P^{sr} are multiplied by the scaling factors X^r (equation (4)) and then, if required, they are scaled down as in equation (7).

(b) If the model is saturated but there is a state with $P^{sr} = 0$ (a chance for probability increase), this particular null probability is replaced with a predicted average increase of P^{sr} in other states computed by the normal method as

above; after this the model is no longer saturated and the iterative scaling of this probability can be continued with the standard algorithm.

It should be noted that in case (b) the new value that replaces the null probability is computed as an average of predicted *actual* increases of probabilities for given repair in other states: these increases will be scaled down with equation (7) because these states, by virtue of the method, will be saturated. As a result, the applied value of the increase will be proportional to the needs of particular situation but, at the same time, it will be additionally constrained.

Figure 4 presents the results obtained after application of such extended procedure to models A and B. In both cases the models can be successfully adjusted in the full examined range, i.e. up to the doubled frequencies of the minor repair. Also, as it can be seen, the final results are virtually identical to the outcomes of the standard (unmodified) procedure in cases when model saturation does not take place (as compared to Fig. 2), while the growth of the probabilities after the saturation point follows the expected course without any instabilities.

4 Conclusions

The purpose of the method presented in this paper is to extend the adjustment algorithm which was proposed in [14] and [16]. The main idea is to modify the model during the iteration by forcing a value greater than zero for a repair probability in situation when this probability reach the limit in other states, i.e. the model saturates. This extension allows to evaluate a class of cases that was not properly handled by the original method.

The proposed approach strives to be as conservative as possible with regard to the amount of alterations introduced to the existing model. While the original method constrains the adjustment operations so that the distribution of the repair probabilities over all deterioration states is not altered, the modification introduced by this extension is more significant and must be applied in a very cautious manner in order to avoid deformation of the model and corruption of the produced results.

In this situation there is a growing need for methods that would evaluate trustworthiness of the generated results. The future work should include development of new metrics that would be able to quantitatively assess modification of the model and to estimate the range of its valid use.

References

[1] Anders, G.J., Endrenyi, J.: Using Life Curves in the Management of Equipment Maintenance. In: PMAPS 2004 Conference, Ames, Iowa (2004)

[2] Anders, G.J., Leite da Silva, A.M.: Cost Related Reliability Measures for Power System Equipment. IEEE Transactions On Power Systems 15(2), 654–660 (2000)

[3] Anders, G.J., Maciejewski, H.: Estimation of impact of maintenance policies on equipment risk of failure. In: Proc. Int. Conf. Dependability of Computer Systems DepCoS – RELCOMEX 2006. IEEE Comp. Soc. Press, Los Alamitos (2006)

[4] Anders, G.J., Sugier, J.: Risk assessment tool for maintenance selection. In: Proc. Int. Conf. Dependability of Computer Systems DepCoS – RELCOMEX 2006. IEEE Comp. Soc. Press, Los Alamitos (2006)
[5] Billinton, R., Allan, R.N.: Reliability Evaluation of Engineering Systems. Plenum Press, London (1983)
[6] Chan, G.K., Asgarpoor, S.: Preventive Maintenance with Markov Processes. In: Proc. 2001 North American Power Symposium, College Station, TX, October 2001, pp. 510–515 (2001)
[7] Endrenyi, J.: Reliability Modeling in Electric Power Systems. Wiley, Chichester (1978)
[8] Endrenyi, J., Anders, G.J., Leite da Silva, A.M.: Probabilistic Evaluation of the Effect of Maintenance on Reliability - An Application. IEEE Transactions on Power Systems 13(2), 575–583 (1998)
[9] Hosseini, M.M., Kerr, R.M., Randall, R.B.: An inspection model with minimal and major maintenance for a system with deterioration and Poisson failures. IEEE Trans. on Reliability 49(1), 88–98 (2008)
[10] Hughes, D.T., Russell, D.S.: Condition Based Risk Management (CBRM), a Vital Step in Investment Planning for Asset Replacement. In: IEE-RTDN Conference, London (2005)
[11] Endrenyi, J.: The Present Status of Maintenance Strategies and the Impact of Maintenance on Reliability. IEEE Trans. Power Systems 16(4), 638–646 (2001)
[12] Limnios, N., Oprisan, G.: Semi-Markov Models and Reliability. Birkhauser, Boston (2001)
[13] Perman, M., Senegacnik, A., Tuma, M.: Semi-Markov Models with an Application to Power-Plant Reliability Analysis. IEEE Transactions on Reliability 46(4), 526–532 (1997)
[14] Sugier, J., Anders, G.J.: Modeling changes in maintenance activities through fine-tuning Markov models of ageing equipment. In: Proc. Int. Conf. Dependability of Computer Systems DepCoS – RELCOMEX 2007. IEEE Comp. Soc. Press, Los Alamitos (2007)
[15] Sugier, J., Anders, G.J.: Verification of Markov models of ageing power equipment. In: Proc. Int. Conf. Probabilistic Methods Applied to Power Systems PMAPS 2008, Rincon, Puerto Rico (2008)
[16] Sugier, J., Anders, G.J.: Modeling Equipment Deterioration for Dependability Analysis. In: Proc. 4th Int. Conf. on Information Technology ICIT 2009, Amman, Jordan (2009)
[17] Sugier, J., Anders, G.J.: Modifying Markov models of ageing equipment for modeling changes in maintenance policies. In: Proc. Int. Conf. Dependability of Computer Systems DepCoS – RELCOMEX 2009. IEEE Comp. Soc. Press, Los Alamitos (2009)
[18] Sugier, J., Anders, G.J.: Modelling equipment deterioration vs. maintenance policy in dependability analysis. In: Al-Dahoud, A. (ed.) Computational intelligence and modern heuristics, In-Teh, Vukovar (2010)
[19] Yin, L., Fricks, R.M., Trivedi, K.S.: Application of Semi-Markov Process and CTMC to Evaluation of UPS System Availability. In: Proc. 2002 Annual Reliability and Maintainability Symposium, pp. 584–591 (2002)
[20] Zhang, T., Nakamura, M., Hatazaki, H.: A decision methodology for maintenance interval of equipment by ordering based on element reparation-replacement rate. In: Power Engineering Society Summer Meeting, vol. 2, pp. 969–974. IEEE, Los Alamitos (2002)

Bad Memory Blocks Exclusion in Linux Operating System

Tomasz Surmacz[1] and Bartosz Zawistowski[2]

[1] Institute of Computers, Control and Robotics,
Wroclaw University of Technology
tomasz.surmacz@pwr.wroc.pl
[2] MSc student at Wroclaw University of Technology
zawistowski.bartosz@gmail.com

Abstract. Memory failures are quite common in todays technology. When they occur, the whole memory bank has to be replaced, even if only few bytes of memory are faulty. With increasing sizes of memory chips the urge not to waste these 'not quite properly working' pieces of equipment becomes bigger and bigger. Operating systems such as Linux already provide mechanisms for memory management which could be utilized to avoid allocating bad memory blocks which have been identified earlier, allowing for a failure-free software operation despite hardware problems. The paper describes problems of detecting memory failures and OS mechanisms that can be used for bad block exclusion. It proposes modifications to Linux kernel allowing a software solution to hardware failures.

1 Introduction

Soft memory errors may be caused by electromagnetic noise and greatly depend on the working environment and the appropriate shielding of the computer system. Cosmic rays are also an important factor, especially in systems with large amounts of system memory or working in vulnerable environments. An estimated error rate of errors caused by cosmic rays is 1 soft error per month per 256 MB of data at sea level [13] (and increasing with height, as the shielding provided by the atmosphere decreases).

Hard errors are defects that are persistent even after rebooting the system. As some studies show [11] not all hardware errors lead to system failures, as some of them may either get cancelled by further data overwriting, or cause silent data corruption. These are the most dangerous to data integrity, as they may remain undetected until a much later time while the system is still running and further data corruption takes place. Also, errors undetected in one of the subsystems may lead to failures appearing in another subsystem [6].

As technology advances, the density of IC elements increases, which allows designing more complex circuits, but also makes harder to produce hardware

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 219–231.
springerlink.com

that is defect-free. Some techniques of dealing with partially defective memory chips have been described in [1] – these include creating mappings for bad bits and storing them in error-free and reliable CMOS, or adding some redundant blocks and permanently blocking the faulty ones at the testing stage. Additional constrains on energy consumption lead to further reduction in currents and charges needed to store a memory cell state. This increases susceptibility of memory cells to random bit flipping from thermal or radiation noise and has to be dealt with at micro- or nanoscale level with appropriate error correction techniques using ECC [1, 10]. These methods however deal with the problem on a circuit design level, trying to provide a view of a reliable and error-free memory chip when seen from the outside. Non-ideal operational environment, overheating or extended exposure to radiation may still cause memory degradation to such an extent, that these techniques fail and some errors start being seen outside. If that happens, the faulty memory chip (or the whole bank) has to be replaced. In Linux, as well as other Unix-like systems, memory management is done through a paging system with the help of a Memory Management Unit (MMU). This allows for a fine-grain exclusion od defective memory regions with a page-size resolution by a Linux kernel with modifications described in this chapter. A system modified this way may be safely run with such partially-faulty memory banks by excluding the faulty regions from system usage.

The rest of this chapter is organized as follows: In sections 2 and 3 we provide short summary of memory errors and testing techniques, in section 4 we discuss memory management in Linux. Our methods of faulty memory exclusion are presented and discussed in sections 5 through 7. In section 8 we summarize the results.

2 Memory Faults and Testing Methods

Memory faults may be combinational or sequential in their nature. Furthermore, these may be either transient or permanent. Combinational faults may be divided in three main categories: stuck-at-0, stuck-at-1 and bridging of two or more lines. They do not depend on the sequence of changes, so generating appropriate set of tests to perform exhaustive testing is easy. In memories, stuck-at faults may either appear in the cells themselves or in the controlling circuitry, e.g. the addressing multiplexers, where they are considered a separate fault category – *address decoder faults*. To test memory cells against the stuck-at errors it is enough to write them with all-ones and all-zeroes patterns and test the resulting values. Bridging of lines and address decoder types of errors require various testing patterns of interleaving zeroes and ones to be written and read from the memory but is still quite straightforward. Some other faults, such as a broken connection between transistors forming a gate or a broken connection between the logic gates can also be modelled and tested as a stuck-at error.

Sequential errors are harder to detect as they manifest themselves only when particular transitions from one state to another take place. As computer memory contains input and output latches, the general sequential fault model has to be applied when constructing test sets.

Functional fault models for memories usually classify faults as *static* (or *simple*) [7], where at most one read or write operation is needed to trigger the fault, and *dynamic*, where a particular sequence of operations is needed for the fault to occur. These can be loosely mapped to combinational and sequential faults, but not as a rule. Furthermore, complex faults may be divided in *linked faults* [7], where some *fault masking* can occur, *single-port* or *multi-port*, and *single-cell* or *multi-cell* (*coupling faults*).

Memory modules without parity checking – known also as *non-parity* modules – are still most common in computer usage. For every physical bit there is exactly one corresponding data bit and no overhead exists. Memory with *parity checking* appends one *parity-bit* to verify correctness of stored data and *non-maskable interrupt* may be triggered that instructs processor to hang up in case of error, so that further data lost can be prevented.

The main disadvantage of parity checking is the lack of error correction mechanisms. It is only possible to check if a memory module is faulty without any opportunity to correct the result of the memory operation, so that further failure-free machine usage could be continued. More sophisticated method is described as ECC [4] (*Error Correction Code*). One of the biggest advantages of ECC is the way the correcting process works – it happens on the fly when errors are being detected. In addition, its implementation is simple and uses only $\log_n N + 1$ control bits for N-bit data input. The cost of applying ECC method is bigger for $N < 32$ and is profitable only for $N > 32$.

The parity generator is also used in ECC but there are several parity bits for data bits. The Hamming distance for coding words is 3 for SEC ECC (*Single-bit Error Correction Code*) so it makes possible to detect and correct only 1-bit errors. A more advanced method – SECDED ECC (*Single-bit Error Correction, Double-bit Error Detection*) allows to correct 1-bit errors too, but it can detect 2-bit errors. It this case, the Hamming distance is 4.

ECC method compared to typical parity checking introduces about 2% speed reduction at average [9].

3 Memory Testing Using Memtest86

Memory faults detection is a complex task, and many testing algorithms have been described in literature. Memory testing can be done either by implementing these algorithms (chosen as best fitting for a particular task) or by using some ready-made software, like Memtest86 [3].

Memtest86 uses *moving inversions* algorithm [2] with several modifications to verify memory modules correctness. It is able to detect two types of faults:

- SAF (*stuck-at faults*),
- AF (*address decoder faults*).

Physical or electrical damages falling in stuck-at category can be easily detected by simple testing algorithms. Bridging, in turn, takes much more tests to be detected, as two or more bits are in the same state independently of written values. Other failures, like faulty addressing lines or a memory that is not present in computer system, are usually easy to detect.

Memtest86 must be run as a standalone program with direct access to memory which may not be obscured by the underlying operating system. Modern Linux distributions include Memtest86 in their bootloader configuration, so when the computer system is started there is a choice to run either the Linux OS or the Memtest86 program (instead of the operating system). When run, the Memtest86 program first builds the memory map by obtaining information from BIOS and analyzing the data provided by ACPI (to skip some reserved locations). It then enters a loop executing a predefined set of tests. If any faulty memory locations are found, they are reported on-screen in a standardized manner, such as:

32-bit address	32-bit mask
`0x03e06e90`	`0xfffffffc`

The address field shows the physical memory address of the beginning of the damaged area and the mask describes which bits are faulty (values set to "1" represent faulty bits). The information given is mostly meant for locating faulty memory chips for the purpose of replacing them. However, the accuracy of identifying addresses and ranges of the faulty areas allows us to use it later to lock the faulty memory pages by kernel and exclude them from the system usage.

4 Memory Management in Linux

Memory in the Linux operating system is organised as follows:

- physical memory is divided in fixed-size pages;
- memory requested by applications is allocated in multiples of page size;
- Linux OS implements a demand-driven memory system [12];
- memory management is supported and supplemented by MMU – if the requested page is not present, it generates page faults and transfers control to the CPU to handle this situation,
- virtual memory provided by the system may be continuous, even though the physical memory does not have to be.

Paging is the default memory management scheme in Linux with typical page size of 4 KB (although Intel processors [8] allow to use 4 MB pages in 32-bit

mode or 2 MB pages in PAE mode). Linux treats pages as the basic unit of memory management. Despite the fact that the smallest addressable unit in processors' word is byte, MMU typically deals with pages. Kernel sources provide several helpful macros that ease the usage of paging in Linux kernel:

- PAGE_SHIFT – determines offset on page (it equals to 12 on x86 machines),
- PAGE_SIZE – defines page size (typically 2^{12} bytes, i.e. 4 KB),
- PAGE_MASK – allows masking some of the offset bits,
- PAGE_PER_PTE, PTRS_PER_PMD, PTRS_PER_PGD – number of entries in Page Table, Middle-level Page Table and Table Directory.

Linux kernel keeps track of every page in order to know its state: what kind of data it contains (kernel code, data structures used by kernel or memory used by user applications) and in how many places it is referenced. The page descriptor is represented by a structure described as follows:

```
struct page {
    /* atomic flags, updated asynchronously */
    page_flags_t flags;
    /* usage counter */
    atomic_t _count;
    /* list of pages, e.g. active_list protected by zone->lru_lock */
    struct list_head lru;
    /* Other fields */
    ...
};
```

There are two fields we should pay attention to:

- `_count` – the usage counter; 0 means that the page is not used, values greater than 0 mean that the page is used by some user processes or by the kernel;
- `flags` – 32-bit or 64-bit number (depending on the kernel version) that describes the status of a page.

All descriptors are stored in `mem_map` table. As the system boots up, all the memory is initially assigned to kernel and one of its main tasks before starting the `init` process and going to multiuser mode is setting up the paging system. This is handled in the `mem_init()` function – it clears unnecessary `PG_reserved` flag in all the pages that are to be returned to the system pool and calculates the total number of pages in the system. Next, it sets appropriate `_count` values for each memory block. Finally, for every page, the `__free_page()` function is called to check if the block is not reserved and if not – decrement the `_count` field, so that the page is returned to the general memory pool and can be used later also by applications running in user-space.

5 Marking Bad Blocks by Linux Kernel

Maintaining proper system behaviour in presence of memory errors requires isolating bad memory blocks and excluding them from further usage. Thanks to the paging system it can be done with granularity of a single page and using some of the existing mechanisms.

The software solution can be basically achieved by two different approaches:

1. A kernel module that detects faulty RAM areas on-the-fly and excludes them from further usage while the system is running.
2. Detection and processing of faulty RAM areas outside of the operating system by an independent testing software and passing appropriate parameters to the operating system.

The first approach requires the whole testing procedure to be known a priori [5]. There are also other problems related to this method:

- every memory access would have to be tested (or checked before the actual operation) in order to eliminate write to a possibly corrupted memory area. This would obviously lead to a significant system slow-down, unless done in hardware,
- testing procedures have to be coded in assembly language in order to eliminate false results while accessing kernel procedures,
- Linux kernel uses as much memory as possible (usually the last 1GB chunk) to provide caching.

The second approach requires stopping the system (i.e. bringing it to a scheduled downtime) in order to check memory for failures, but is universal and can be applied to different computer architectures (testing only relies on other software used in this procedure). Also, testing performed in this mode can be exhaustive and may take as much time as needed to fully and thoroughly test the memory before proceeding with normal system operation.

Availability of Linux kernel source code gives possibility to apply additional fixes to source code wherever they are needed. As it is an open source operating system, we are able to download kernel sources and extend the current memory management subsystem. The modified OS is then able to run on a partially faulty memory hardware as long as some steps are taken beforehand. Memory faults have to be identified first (they can span across several kilobytes of continuous memory address space), but disabling the broken addresses permits the whole system to run stable and without any data corruption.

The following steps have to be taken manually in order to exclude bad blocks from user space usage:

- Save (or write down) all faulty regions from *memtest86* output,
- pass them to the Linux kernel during the booting process,

Disabling faulty addresses inside a kernel is then performed in two steps:

1. Normalize the given addresses with their masks to get a pool of pages that should be marked as locked,
2. exclude them from memory pool by marking the pages as used by the kernel with a special flag, so they will not be reclaimed later.

Parameters passed to grub or LILO are in fact interpreted in the monolithic part of the kernel that starts as the first part of the booting process, so any extensions to parameters syntax have to be actually done in kernel sources, not in grub. If some other bootstrap loader is used (other than grub or LILO), it may be problematic to pass any parameters to kernel at all. In such case the only possibility is to recompile the kernel with the predefined code for exclusion of some predefined memory areas.

Linux kernels (and also the boot loaders such as grub or LILO) impose limits on the size of the parameters that may be passed through them and that limit is set to 255 characters. If the memory faults are numerous, but sparse, some of the information gathered from the *memtest86* output has to be aggregated. In extreme cases, when the faulty block list would still exceed the parameter limits, this may lead to a need of covering also a range of properly working memory addresses just in order to fit in the parameters length limit.

Due to the fact that *address/mask* pairs describe some memory ranges (several faulty regions can be described as faulty by this method), every pair has to be normalised first in order to find addresses of all affected memory pages. This is done while the kernel boots up. By applying two simple bitwise operations it is possible to get a page-aligned starting addresses of the faulty memory block:

```
mod_mask |= ~PAGE_MASK;
mod_addr &= mod_mask;
```

After this normalisation we are able to obtain all other addresses belonging to the faulty range by iterating over all other addresses using the same mask. If it turns out that addresses do not belong to the same memory page – the next one has to be taken into account and be marked as bad too. The modified code executed by the kernel when the memory system initialization is taking place is shown in fig. 1.

While the kernel is booting up, it calculates the number of pages by obtaining the size of memory installed in computer system. When additional information is passed to kernel it marks pages with particular flags so that in further initialization they are used in special manner. In order to exclude faulty blocks from usage, the reference counter cannot be decremented to 0 (as it applies to all pages used by kernel), as the page could be later reclaimed for some other use. It is possible however to extend the paging system by adding some custom-defined flags to the ones already defined

```
unsigned long a; /* address */
unsigned long m; /* mask */
unsigned long oa = a;
unsigned long r = (a | m) + 1;
if(!r) {
    return 0; /* overflow? */
}
r = (r & ~m) | (a & m);
if(r < oa) {
    return 0;  /* mark the whole area as bad */
}
return r;      /* next address found */
```

Fig. 1 Modified memory initialization in `mem_init()` kernel function

in `include/linux/page-flags.h`. We have introduced an extra flag called `PG_memlocked`, as show in fig. 2.

```
enum pageflags {
  PG_locked,
  PG_error,
  PG_referenced,
  PG_uptodate,
  PG_dirty,
...
  PG_memlocked /* new page state */
};
```

Fig. 2 Extra flag defined in include/linux/page-flags.h

This flag is used to mark all faulty pages as the system boots up and prevents the `__free_page()` function run at the end of the `mem_init()` procedure from returning the page to the general memory pool. As the locked-out page is not going to be used by any process, it would be a very likely candidate for swapping out. But swapping for a locked-out page would effectively mean that it gets reused and allocated to some other process, while the unreferenced, but locked-out page is permanently moved to the swap space. So swapping for locked-out pages has to be prevented and the simplest method it to keep them as kernel-allocated pages (by setting the `PG_locked` flag in addition to `PG_memlocked`), as kernel pages are never swapped-out.

Address/mask pairs can be passed during booting process from boot loader like *grub* or can be compiled into kernel by checking *Built-in kernel command line* option during kernel configuration for custom build. Both methods may require increasing the value of `COMMAND_LINE_SIZE` defined in `include/asm-generic/setup.h` file.

6 Testing Methodology

Testing the operating system stability during typical system usage is problematic as several processes are running concurrently to provide different services needed for normal system operation. The set of running processes should be minimised to facilitate easier isolation and replication of problems. There are two basic ways of detecting the system being unstable due to the memory corruption:

- Kernel crashes while obtaining more memory, e.g. while loading a device driver,
- Application that allocates memory and writes a predefined pattern does not get the same data during the read operation.

The first case results in system panic that leaves traces in appropriate log files, which can be checked by executing `cat /var/log/messages /var/log/kern.log | grep panic`. The second case cannot be easily detected under normal system operation, but requires some testing software to be run. We can distinguish two kernel failures:

- Hard panic ("*Aieee!*") – only a system reboot is possible to recover from failure,
- Soft panic ("*Oops*") – system can be still used but a particular operation did not succeed.

Hard panics happen usually in interrupt handling routines of the drivers, basically because of a null pointer dereference. Later, the device driver cannot handle incoming interrupts and causes the kernel panic. Soft panics happen outside of interrupt handling code and allow the operating system to continue, but without the crashed driver. Hard panics are signalled by blinking keyboard LEDs and freezed screen output. No input can be handled by the system and only a hard reset is possible. The stack trace is rarely logged to `/var/log/messages` file and only the console screen dump can be useful to get some information about the failure. Soft panics may provide more useful information, but some effort is required beforehand – "`Kernel Hacking -> Detect Soft Lockups`" option has to be enabled *a priori* in order to get all debug messages during system crash. Writing crash info to a file may still not be possible (due to a crashed driver or just because of inability to "sync", i.e. commit the memory-buffered write operations to disk), so it is also beneficial to set-up a serial console. This can be done by enabling "`Device Drivers -> Character devices -> Serial drivers -> Console on 8250/16550`" and appending `console=ttyS0,115200` to *grub* or *LILO*. Lastly, to log the stack trace of drivers working in non-interrupt-driven mode, "early printk" has to be enabled by selecting "`Kernel Hacking -> Early printk`" in `.config` file stored in root of Linux kernel source directory.

To test user-space memory for possible corruption within a running system we run a test application that allocates as much memory as possible and performs multiple write/read tests. There is however a limit on the amount of memory that can be loaded by an application without getting a `SIGKILL` signal. The smallest chunk that can be allocated is the page size defined as `PAGE_SIZE` constant. For this reason there is no need to call `malloc()` with smaller values than 4 KB for typical 32-bit architecture. To make the solution universal we call `getpagesize()` system function, which returns the page size of the operating system. Similar results can be achieved by `getconf PAGESIZE` command in shell command prompt or `sysconf(_SC_PAGESIZE)` function. We use the first approach. Figure 3 demonstrates how much memory can be allocated by a program executed in user-space.

```
#include <unistd.h>
#include <stdlib.h>

int page_size = getpagesize();

int i = 1 ;
void *p;
for (;;) {
  p = malloc(page_size * i);
  if (p==NULL)
    break ;
  free(ptr);
  ++i;
}
/* (i - 1) now stores the number of pages */
return (i-1);
```

Fig. 3 Code that calculates the number of pages that can be allocated by a user process

After calculating the maximum number of pages that can be allocated, each fixed size chunk has to be tested with write/read patterns to find possible inconsistencies.

The overall testing process can be described as follows:

1. Check all necessary logging options in kernel configuration before compiling the kernel.
2. Modify bootolader configuration file to enable serial console.
3. Boot up the system with new kernel in single-user mode.
4. Disable swaping by calling `swapoff` or by editing `/etc/fstab` file and modifying the entry related to swap partition so that it will not be mounted.

5. Calculate the number of pages that can be allocated by a user-space program.
6. For each test pattern check allocated memory by `assert()` macro.
7. Load all the drivers as modules and check for hard kernel panics.
8. Check for "panic" entries in syslogd files.

7 Limitations and Further Perspective

Presented solution does not require any additional hardware and is only based on a software kernel extension. Typical booting process for an x86/i86 architecture personal computer is a quite clearly described process. At the very early stage, the processor operates in real mode and only 640KB of memory can be used. Kernel loading can be divided in two stages:

1. A smaller piece of code that is loaded somewhere below the first 640 kilobytes of memory, which is often called as bootstrap.
2. A bigger part loaded above 1 MB of memory where kernel operates in protected mode.

In addition to that, some kernel modules may be loaded at later time, triggered by configuration options or a hardware-detection code that gets executed during the system startup. The critical part of the booting process happens however in the mentioned two early stages, as both of them need to be loaded in memory parts that are not faulty. The kernel code is loaded unconditionally without a choice of preferred memory addresses to use, and if some pages happen to be faulty, the results may be unpredictable. High level initialization takes place in `start_kernel()` procedure where memory management is being set up and then faulty regions may be mapped to pages. Kernel modules pose no problem, as they are loaded after the memory system is initialized and the faulty pages are already mapped-out.

If the memory chips are damaged in such a way that the early-stage kernel code is loaded into faulty pages, the only solution may be to change the system memory by manually swapping the physical memory banks (i.e. placing them in different memory slots so thet their physical addresses are arranged differently) ore use some fault-free banks for the physical addresses used by the bootloader. In typical systems it is the last gigabyte of physical memory.

Another issue may appear in systems where page size is bigger than typical 4 KB. If we consider a 4 MB page and huge amount of faulty addresses spread out on not consecutive areas, then even small and sparse errors may cause exclusion of large address ranges from further use. Such systems are however in the experimental stage of deployment, and until they are widely used, we may investigate other solutions that will access and exclude memory blocks on a sub-page level.

The current solution has been applied and tested on early versions of 2.6 kernel series. Several data structures have been changed since then – for

instance, the `page_flags_t` type has been changed to `unsigned long` and some other additional memory features have been added to the latest kernel versions.

Current versions of the memory testing software do not support dumping the test output to a filesystem installed on a device (i.e. the hard drive or a USB memory stick), so it may be quite difficult and troublesome to save all the addresses that are produced on screen by the testing procedure. Memtest86 operates in real mode so other extensions may not be easy to implement, it is however one of the concerns for further development, as it would greatly simplify the automatization of the whole process.

Buffer size for command line parameters is around 256B so it may be impossible to pass all faulty addresses to Linux kernel during boot. In such case, the kernel has to be modified to include a `mem_init()` function extension with a predefined list of pages to exclude. It can be done by hardcoding a table of `unsigend long` *address/mask* pairs that will be used in addition of the parameters passed though the bootloader, This approach requires the kernel to be recompiled every time the pool of faulty regions changes, which may be tricky on a system that must run with these errors, so it must either be done by a kernel compilation performed on a different machine, or by a temporary addition of new faulty memory addresses through the bootloader parameters. This way that the system may run stable while the kernel is recompiled with an updated map of faulty memory locations. The preparation of such a system setup may be a tricky one, even though the proposed solution allows for a safe and complete exclusion of faulty memory blocks.

8 Conclusions

Linux kernel modifications described in this chapter allow fault-free operation of the Linux OS on a hardware where memory banks are partially faulty but the faulty addresses may be identified by some external testing programs. Faulty memory regions are excluded from system usage with page-size granularity by locking them as unswappable kernel-used memory, without actually accessing them for any purpose.

The described modifications have been implemented and tested in 2.6 series of kernels, up to version 2.6.6, and the implementation effort is now directed to porting them to current versions, for both the 32- and 64-bit systems.

References

[1] Biswas, S., Metodi, T., Chong, F., Kastner, R.: A pageable, defect-tolerant nanoscale memory system. In: IEEE International Symposium on Nanoscale Architectures, NANOSARCH 2007, pp. 85–92 (2007), doi:10.1109/NANOARCH.2007.4400862

[2] De Jonge, J.H., Smeulders, A.J.: Moving Inversions Test Pattern is Thorough, Yet Speedy. International Computer Design (1976)

[3] Demeulemeester, S.: Memtest86, an Advanced Memory Diagnostic Tool (2010), http://www.memtest.org
[4] Elkind, S., Siewiorek, D.: Reliability and performance of error-correcting memory and register arrays. IEEE Transactions on Computers C-29(10), 920–927 (1980), doi:10.1109/TC.1980.1675475
[5] Elm, C., Klein, M., Tavangarian, D.: Automatic On-line Memory Tests in Workstations. In: Records of the IEEE International Workshop on Memory Technology, Design and Testing (1994)
[6] Gu, W., Kalbarczyk, Z., Ravishankar, Iyer, K., Yang, Z.: Characterization of Linux kernel behavior under errors. In: Proceedings of 2003 International Conference on Dependable Systems and Networks, pp. 459–468 (2003), doi:10.1109/DSN.2003.1209956
[7] Hamdioui, S.: Testing static random access memories: defects, fault models, and test pattern. Kluwer Academic Publishers, Dordrecht (2004)
[8] Intel Corporation IA-32 Intel Architecture Software Developer's Manual. In: System Programming Guide, vol. 3 (2009), http://www.intel.com/products/processor/manuals/
[9] Kozierok, C.: The PC Guide – Memory Errors, Detection and Correction (2001)
[10] Maestro, J., Reviriego, P.: Selection of the optimal memory configuration in a system affected by soft errors. IEEE Transactions on Device and Materials Reliability 9(3), 403–411 (2009), doi:10.1109/TDMR.2009.2023081
[11] Messer, A., Bernadat, P., Fu, G., Chen, D., Dimitrijevic, Z., Lie, D., Mannaru, D., Riska, A., Milojicic, D.: Susceptibility of commodity systems and software to memory soft errors. IEEE Transactions on Computers 53(12), 1557–1568 (2004), doi:10.1109/TC.2004.119
[12] Silberschats, A., Peterson, J.L., Galvin, P.B.: Operating System Concepts. Addison-Wesley Publishing Company, Inc., Reading (1991)
[13] Swift, W.: Memory errors: roll the dice! IEEE Antennas and Propagation Magazine 38(6), 124–125 (1996), doi:10.1109/MAP.1996.556530

Metamodel and UML Profile for Functional Programming Languages

Marcin Szlenk

Warsaw University of Technology, Institute of Control & Computation Engineering,
Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: m.szlenk@ia.pw.edu.pl

Abstract. Functional programming languages are ideally suited for developing dependable software, but not much work have been done on modeling functional programs. Although UML is mainly based on concepts which are native to imperative object-oriented programming languages, this chapter shows how – through the profile mechanism – it can be used to model software that is to be implemented in a functional programming language. In this chapter Haskell was chosen as one of the most popular modern, pure functional languages. First, a partial metamodel of Haskell is defined and then the corresponding UML profile is presented.

1 Introduction

Unified Modeling Language (UML) [11, 14] is intended to be a universal general-purpose modeling language for software systems. UML contains an extensibility capability for customizing models for particular domains or platforms, where UML extensions are organized into *profiles*. Basic UML – without profiles – reflects the imperative and object-oriented paradigm. A system is modeled as a collection of discrete objects that interact to perform given work. Using UML to model software that is to be implemented in languages supporting other programming paradigms may seem to be odd at first, however, creating a dedicated UML profile may give a natural and convenient modeling notation, which at the same time benefits from the existing tool support for UML. This chapter is an attempt at defining the UML profile for a programming language built on a functional programming paradigm.

2 Functional Programming

Functional programming treats computations as the evaluation of functions (or expressions), avoiding using state and mutable data [1, 4]. Thus, functions are stated in a declarative way, where in contrast to the imperative programming, a function definition shows what is to be done, rather than how it is to be done in terms of state changes. Functions are treated here as any other values, that is, they

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 233–242.
springerlink.com

can be passed as arguments to other functions or be returned as a result of a function. Some functional programming languages, e.g. ML variants like Standard ML [8], Objective Caml [13] or F# [3], allow to program in both functional and imperative (including object-oriented) style, while the others, e.g. Miranda [16] or Haskell [5, 6], lack imperative programming constructs and remain *purely* functional.

As far as UML modeling is concerned, one can distinguish two types of models: dynamic and static. The dynamic model is used to express the behaviour of a system over time, whereas the static model shows those aspects that do not change over time. The dynamism is intuitively understood here as changes in a system state (which is constituted of the states of its objects), however, in pure functional programs there is no concept of state or order of execution. It is left up to the runtime system how to compute the values given the relations to be satisfied between them. In that sense, in a functional programming language (or at least in its pure subset) only the static structure of program is explicitly specified, while the dynamic aspects remain hidden.

2.1 Haskell

Haskell is nowadays probably the most popular purely functional programming language. It has been designed as a vehicle for functional programming teaching, research, and applications and efforts in improving it are still ongoing. In this chapter the current Haskell specification [6] is used.

```
module AStack (Stack, push, pop, top, size) where

data Stack a = Empty | MkStack a (Stack a)

push :: a -> Stack a -> Stack a
push x s = MkStack x s

size :: Stack a -> Int
size s = length (stkToLst s) where
            stkToLst  Empty         = []
            stkToLst (MkStack x s)  = x : stkToLst s

pop :: Stack a -> (a, Stack a)
pop (MkStack x s) = (x, s)

top :: Stack a -> a
top (MkStack x s) = x
```

Fig. 1 A sample Haskell program

In Fig. 1 a sample program written in Haskell is presented. This is a simplified example taken from [6]. The program is organized into one module called '`AStack`', which contains a user-defined *algebraic data type* '`Stack`' (parameterized with a *type variable* '`a`') and functions '`push`', '`size`', '`pop`', and '`top`' for typical stack operations. The type variable '`a`' can be replaced by concrete types (such as '`Int`', '`Float`', and so on) when a given value of type '`Stack`' is declared or the functions are applied. This serves as a parametric polymorphism mechanism.[1] Both the data type and the functions are explicitly (they appear on an *export list*) exported by the module and are available to anyone importing the module. This sample program will be used later to explain and present the application of a proposed UML profile, however, the basic knowledge of functional programming is assumed.

3 Metamodel

Although Haskell contains some unique features, this chapter will stick to a subset which is common to several other functional programming languages. In particular, the system of *type classes* [6] will be omitted. It not only simplifies further consideration but also allows to easily adapt the proposed profile to other functional languages.

The strategy of defining the UML profile for Haskell is here similar to the one used in UML profile specifications provided by Object Management Group (OMG), e.g. [10]. First, the Haskell *metamodel* (a model of Haskell expressed in UML) is defined. The goal of defining this metamodel is to set the scope of Haskell language which will be included in the profile and to set the level of abstraction for the profile elements. The metamodel presented below is intended to provide sufficient details to create Haskell design and implementation models.[2] The assumed level of abstraction allows to partially generate Haskell code (that will need to be further completed by hand), but does not allow for full code generation from models. As stated before, it is not a complete metamodel of the Haskell language yet it describes a consistent and useful subset of the language. The UML profile corresponding to this metamodel will be then defined in Sect. 4 Profile.

3.1 Haskell Metamodel

The Haskell metamodel is presented as three class diagrams completed with a description of the important features of the diagram and additional constraints expressed in Object Constraint Language (OCL) [7].

---

[1] The other kind of polymorphism called *overloading* can be defined in Haskell using *type classes*.

[2] They correspond to Platform Specific Models (PSM) in Model Driven Architecture (MDA) approach [9].

Module Contents (Fig. 2). Haskell programs are organized into *modules*, which play a similar role to packages in Java or namespaces in C++ language. Two kinds of basic program elements defined in such a module are *functions* and *user data types*, but it is not obligatory to define them in a certain module, i.e. the '`module Name where`' header at the beginning of a file (see Fig. 1) can be omitted.[3] Before the '`where`' keyword a parenthesized list of functions, types and constructors exported by a module can be added. The attribute '`isExported`' in the metamodel indicates whether the element appears on the export list of a module. Modules can also import other modules (their exported elements) by adding the '`import`' declarations at the beginning of the module.[4] In fact, the module system in Haskell allows also for importing chosen elements of modules and hiding others. Module imports may not form a cycle. Note that no additional constraints in the metamodel are needed here because this fact results from the semantics of the aggregation relationship in UML, which is transitive and antisymmetric [11].

Haskell functions take zero or more arguments and must always return a result. A zero-argument function is called a *value*. Haskell is a statically and strongly typed language, but the user does not have to explicitly specify the types of functions as they can be inferred by the system. Functions in Haskell are pure, i.e. they do not have any *side-effects*. To examine and modify the current state of the world, e.g. read and write files, read from a keyboard or print something on a screen, one has to use IO (input/output) *actions*. Every IO action returns a value, but in the type system the returned value is tagged with '`IO`' type, distinguishing actions from functions. For example, the type of the function '`getChar`' is:

```
getChar :: IO Char,
```

what means that this function is actually an action and when it is invoked, the result will have type '`Char`'. IO actions can be passed to functions. The attribute '`isIO`' in the metamodel indicates whether the type is an '`IO`' type.

User data types are defined using '`data`' keyword (see Fig. 1). They are algebraic types, i.e. any value of such a type is created using a *constructor*, which is just a function, expecting some arguments (of other types) and delivering a value of the given user type. A constructor may also not take any arguments (may be a value) and an algebraic type may have many constructors (these are separated with the '`|`' character). A constructor cannot be an action and its result type is the user type whose values it constructs. This constraint can be expressed in OCL as below:

```
Constructor
resultType = userType and resultType.isIO = False.
```

Types (Fig. 3). The most frequently used basic types in Haskell are: '`Bool`', '`Char`', '`String`', '`Int`', '`Integer`' (infinite-precision integers), '`Float`', '`Double`', and the *unit type* '`()`' (which is used when an IO action returns nothing).

[3] In this case, the header is assumed to be '`module Main (main) where`'.

[4] In fact, the module system in Haskell allows also for importing chosen elements of modules and hiding others.

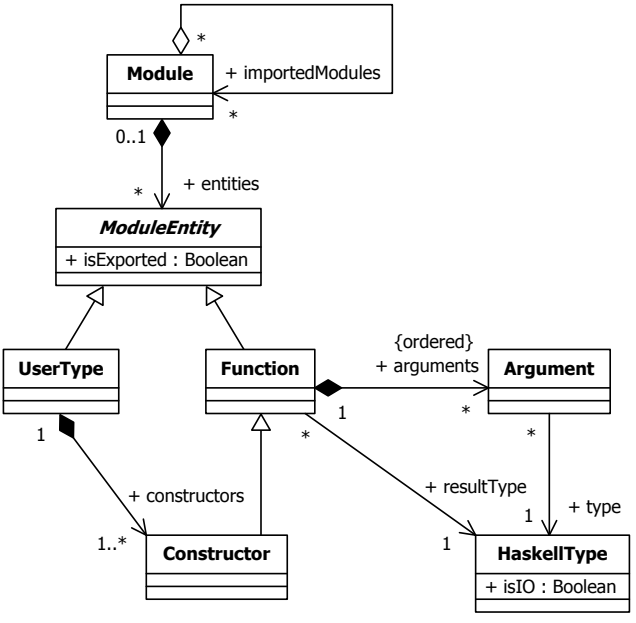


Fig. 2 Module contents

More complex types, like e.g. list types, are constructed from other types and are shown in Fig. 4. Polymorphic types are described in Haskell using type variables. For example, the type variable '`a`' in Fig. 1 represents any type. A user-defined type (an algebraic type) can be parameterized with one or more type variables and thus become a polymorphic one.[5]

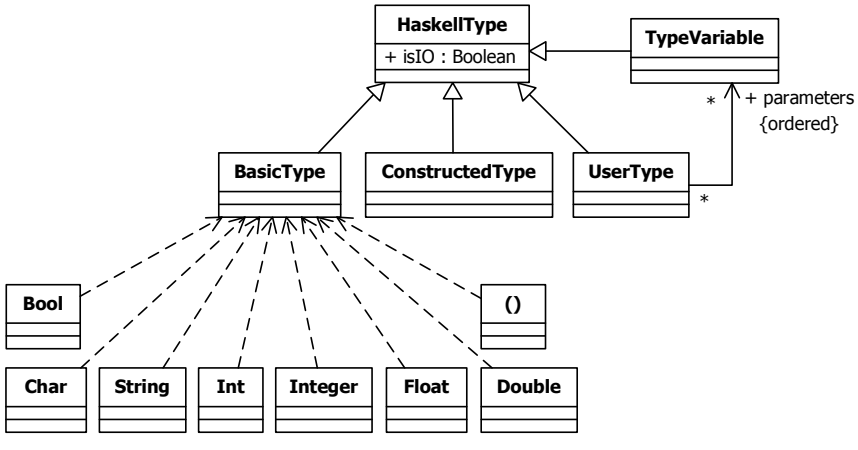


Fig. 3 Types

[5] In fact, Haskell's type system is more sophisticated, but the simplified description presented here seems adequate to its purpose.

Constructed types (Fig. 4). Haskell offers also types which are constructed from other types (which themselves can be basic or constructed). These constructed types are:

- *list types* (e.g. '`[Char]`' is a list of characters),
- *tuple types* (e.g. '`(Int, Float)`' is an ordered pair, where the first element is an integer and the second is a real), and
- *function types* (e.g. '`Char -> Bool`' is a function which takes a character and returns a boolean result).

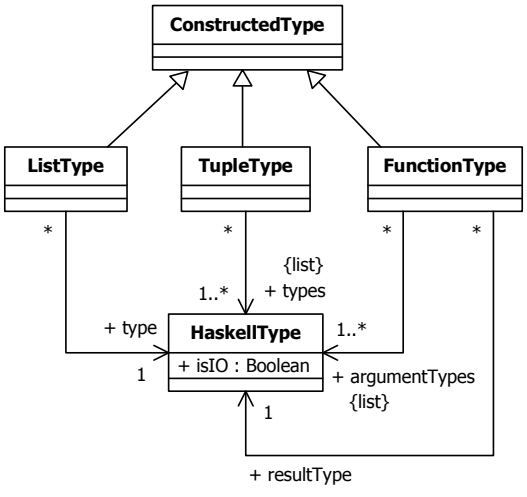


Fig. 4 Constructed types

Lists can hold an arbitrary number of elements, but these elements must all be of the same type. This contrasts with tuples, which hold only a fixed number of elements, but can be heterogeneous.

4 Profile

UML can be tailored to specific domains or programming environments by defining its dialect as a profile [14]. A UML profile identifies a subset of UML and defines *stereotypes* and constraints that can be applied to the selected UML subset. This section presents a UML profile for Haskell, but the profile presented can be also adopted to other functional languages. The profile consists of eight stereotypes which are a direct mapping of the concepts defined in the Haskell metamodel presented in the previous section. Table 1 depicts the relation between the stereotypes from the profile and the Haskell matamodel, as well as UML base elements for the stereotypes (i.e. elements to which the stereotypes can be applied).

Table 1 Mapping metamodel concepts to profile elements

Metamodel element	Stereotype	UML base element
Module	«Module»	Class
Function	«Function»	Operation
Function	«Value»	Attribute
Function	«IOAction»	Operation
UserType	«UserType»	Class or Parameterized class
Constructor	«Constructor»	Operation
Module.entities	«Contents»	Dependency
Module.importedModules	«Import»	Dependency

4.1 Stereotypes

In the following, the stereotypes in the profile and their use are briefly described.

Module. This stereotype can be applied to a Class. Classes annotated with this stereotype represent Haskell modules. Functions defined in a given module can be then specified on an operation list of a Class or on an attribute list if a function is a value.

Function. This stereotype is used on Operations to represent pure functions defined in a Haskell program. The default UML syntax for an operation is used:

```
name (parameter: parameter-type, ): return-type,
```

where '`name`' is the name of the given function, '`parameter`' is the name of the function argument, '`parameter-type`' is the name of the type of that argument, and '`return-type`' is the name of the type of the function result. Both the names of arguments, the names of their types and the name of the result type are optional (their appearance depends on how detailed the function is modeled). The name of the argument type and the name of the result type can be any Haskell type expressions. The only difference is for parameterized types, where the names of the type variables should be enclosed in angle brackets (`< >`), similar to the UML notation for *parameterized classes* (template classes) [14]. For example, '`Stack a`' and '`Either a b`' should be written as '`Stack<a>`' and '`Either<a,b>`', respectively. This is consistent to the way parameterized user-defined types are modeled (see the description for the *UserType* stereotype below).

Value. This stereotype should be used to show a zero-argument pure function and it can be applied to an Attribute. The default UML syntax for an attribute is used:

```
name: type = value,
```

where ‘`name`’ is the name of the given value, ‘`type`’ is the name of the value type and ‘`value`’ is the given value. Only the name of the value is obligatory. The syntax rules for the name of the value type are the same like in the case of the *Function* stereotype.

IOAction. This stereotype is used on Operations to represent IO actions. Its use is the same as of the *Function* stereotype. Zero-argument action should be also shown as an operation with *IOAction* stereotype. The operations annotated with the *Function* or *IOAction* stereotypes and the attributes with the *Value* stereotype can be declared only in a class having the *Module* stereotype.

UserType. User-defined types should be shown as Classes annotated with a *UserType* stereotype. Constructors of such a type can be then specified on an operation list of a Class. For user-defined types that are parameterized with type variables the *UserType* stereotype should be applied to parameterized classes, where the number and the names of the parameters correspond to the number and the names of the type variables.

Constructor. This stereotype can be applied to an Operation. Operations annotated with this stereotype represent constructors of a given user data type. The syntax for such an operation is the same as in the case of the *Function* stereotype. The only difference is that the arguments of the constructor do not have names. The operations annotated with the *Constructor* stereotype can be only declared in a class bearing the *UserType* stereotype.

Contents. Functions defined in a module are specified on an operation list of a class representing this module. To show that a given user type is defined in a given module one should use a UML Dependency relationship with a *Contents* stereotype applied to it. This relationship should connect a class representing the module to a class representing the user type. The given user type may be contained in only one module.

Import. This stereotype is to be applied to a Dependency relationship connecting two classes representing modules where one of these modules imports the other. This relationship should go from the class representing the importing module to the class representing the imported module. The information whether a module exports a function or data type defined in it should be shown as UML *visibility markers* [14] placed before the name of a function or data type. The marker ‘+’ (public) denotes that the module element is exported and ‘–’ (private) that it is not exported.

In Fig. 5 a model of a sample Haskell program from Fig. 1 is presented showing the application of some of the stereotypes.

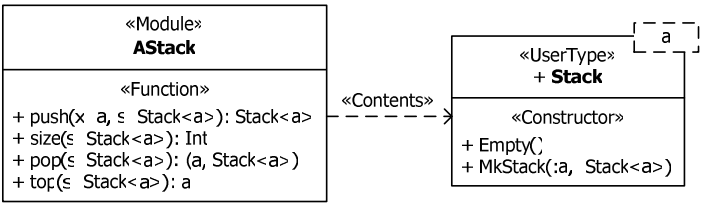


Fig. 5 A sample model

4.2 Constraints

A stereotyped UML element may have additional constraints beyond those of the base element [14]. Some of the additional constraints have been stated above, e.g. that the user type may be contained in only one module, and some other are omitted here. All such constraints come directly from the Haskell metamodel, what is an essential advantage of creating the metamodel of the language for which the UML profile is being defined.

5 Related Work

It seems that, so far, no work has been done on tailoring UML to model functional programs. In [17], rather than tailor UML to model Haskell programs, the translation from standard UML elements to Haskell is proposed. As the author himself admits, it results in some awkwardness in converting from the object-oriented to the functional paradigm and the Haskell code produced this way looks much more imperative than functional.

In general, not much work seems to have been done on modeling functional programs. In [15] a graphical modeling language is proposed, however, it is not related to UML in any way. Some *visual functional programming languages* [2, 12] have been also defined, but they focus on graphical representation of algorithms rather than abstract models of programs.

6 Conclusion and Further Work

The main idea behind this chapter is filling the gap in the area of graphical notations for modeling functional programs. From the practical point of view, it seems attractive to use a widely known UML notation with its extensive tool support, rather than define a new notation from scratch. For that reason, the work on defining a metamodel and a UML profile for the Haskell language has been undertaken. Some of the initial results of this work have been presented in this chapter. Further work will focus on broadening the scope of Haskell included in the profile, as well as on providing metamodel and profile implementations for popular UML modeling tools.

References

[1] Backus, J.: Can Programming Be Liberated from the von Neumann Style?A Functional Style and Its Algebra of Programs. Communications of the ACM 21(8), 613–641 (1978)
[2] Cardelli, L.: Two-dimensional syntax for functional languages. In: Proceedings of ECICS, vol. 82, pp. 139–151 (1983)
[3] Harrop, J.: F# for Scientists. Wiley Interscience, Hoboken (2008)
[4] Hudak, P.: Conception, Evolution, and Application of Functional Programming Languages. ACM Computing Surveys 21(3), 359–411 (1989)
[5] Hutton, G.: Programming in Haskell. Cambridge University Press, Cambridge (2007)
[6] Jones, S.P.: Haskell 98 Language and Libraries: The Revised Report. Cambridge University Press, Cambridge (2003)
[7] Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling With UML. Addison-Wesley, Reading (1998)
[8] Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML (Revised). MIT Press, Cambridge (1997)
[9] Object Management Group, MDA Guide Version 1.0.1 (omg/03-06-01) (2003)
[10] Object Management Group, Metamodel and UML Profile for Java and EJB Specification (formal/04-02-02) (2004)
[11] Object Management Group UML 2.3 Superstructure Specification (formal/2010-05-05) (2010)
[12] Reekie, H.J .: Realtime Signal Processing: Dataflow, Visual, and Functional Programming. PhD thesis, University of Technology at Sydney (1995)
[13] Rémy, D.: Using, Understanding, and Unraveling the OCaml Language. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.) APPSEM 2000. LNCS, vol. 2395, pp. 413–537. Springer, Heidelberg (2002)
[14] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, 2nd edn. Addison-Wesley, Reading (2004)
[15] Russell, D.: FAD: A Functional Analysis and Design Methodology. PhD thesis, University of Kent at Canterbury (2001)
[16] Turner, D.A.: Miranda: A non-strict functional language with polymorphic types. In: Jouannaud, J.-P. (ed.) FPCA 1985. LNCS, vol. 201, pp. 1–16. Springer, Heidelberg (1985)
[17] Wakeling, D.: A Design Methodology for Functional Programs. In: Taha, W. (ed.) SAIG 2001. LNCS, vol. 2196, pp. 146–161. Springer, Heidelberg (2001)

Resource Co-allocation Algorithms for Job Batch Scheduling in Dependable Distributed Computing

Victor Toporkov[1], Dmitry Yemelyanov[1], Anna Toporkova[2], and Alexander Bobchenkov[1]

[1] Moscow Power Engineering Institute (Technical University), ul. Krasnokazarmennaya 14, Moscow, 111250 Russia
e-mail: ToporkovVV@mpei.ru, {groddenator,yemelyanov.dmitry}@gmail.com
[2] Moscow State Institute of Electronics and Mathematics (Technical University), Bolshoy Trekhsvyatitelsky per. 1-3/12, Moscow, 109028 Russia
e-mail: annastan@mail.ru

Abstract. This work presents slot selection algorithms in economic models for independent job batch scheduling in distributed computing with non-dedicated resources. Existing approaches towards resource co-allocation and multiprocessor job scheduling in economic models of distributed computing are based on search of time-slots in resource occupancy schedules. The sought time-slots must match requirements of necessary span, computational resource properties, and cost. Usually such scheduling methods consider only one suited variant of time-slot set. This work discloses a scheduling scheme that features multi-variant search. Two algorithms of linear complexity for search of alternative variants are proposed and compared. Having several optional resource configurations for each job makes an opportunity to perform an optimization of execution of the whole batch of jobs and to increase overall efficiency of scheduling.

1 Introduction

Job control is among the most difficult problems in the enterprise of distributed computing in the case of non-dedicated resources that are shared with their owners. One must take into account the heterogeneity, changing composition, different owners of different nodes, and the scale of the computing environment. Economic models of scheduling are based on the concept of fair resource distribution between users and owners of computational nodes. They are effectively used in such spheres of distributed computing as Grid [1], cloud computing [2], and multiagent systems [3].

Two lasting trends can be distinguished among various approaches to the organization of computations in distributed environments [4-6]. One of them is based on the use of available resources when the role of mediator between users and computation nodes is played by special application agents called brokers [7, 8]. The other trend is closely related to the creation of virtual organizations (VO) [4]; it is mainly oriented to grid systems [4-6].

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 243–256.
springerlink.com

Both approaches have certain advantages and disadvantages. The resource management systems based on the first approach are well scalable and can be adapted to specific features of various applications. Resource brokers usually implement some economic policy in accordance with the application-level scheduling concept [7, 8]. However, the use of various optimization criteria of job execution by independent users (when jobs may compete with each other) can deteriorate such integral characteristics as total execution time of a batch of jobs and resource utilization. The creation of VO naturally restricts the scalability of job control systems. Nevertheless the use of certain rules for allocation and consumption of resources makes it possible to improve the efficiency of resource planning and allocation at the level of job flows [5]. The corresponding functions are implemented within a hierarchical structure consisting of a metascheduler and subordinate job schedulers [4-6] that are controlled by the metascheduler and in turn interact with resource managers (e.g., with batch job processing systems). The set of specific VO rules allows overall increase in the quality of service (QoS) for jobs and resource usage efficiency. It is worth noting that both approaches assume that applications are scheduled based on dynamically changing information about the global environment; both approaches make it possible to implement various resource management scenarios. Hence, we may speak not only of a scheduling algorithm but rather of a scheduling strategy, that is, of a combination of various methods of external and local scheduling, data allocation methods etc. [9, 10].

The model proposed in [11] is based on the concept of fair resource distribution between users and owners of computational nodes by means of economic mechanisms in VO. Existing approaches towards resource co-allocation and multiprocessor job scheduling in economic models of distributed computing are based on search of time-slots in resource occupancy schedules. The sought time-slots must match requirements of necessary span, computational resource properties, and cost [7, 8, 11]. There is the description of some approaches to forming of different deadline and budget constrained strategies of scheduling in [7]. Heuristic algorithms for slot selection based on user defined utility functions are introduced in [8]. Usually economic scheduling techniques consider only one suited variant of time-slot set.

In this work, a scheduling scheme with multi-variant slot search is proposed. Having several optional resource configurations for each job makes an opportunity to perform an optimization of execution of the whole batch of jobs and to increase overall efficiency of scheduling and QoS. We propose two algorithms for slot selection that feature linear complexity $O(m)$, where m is the number of available time-slots.

Existing slot search algorithms, such as backfilling [12], do not support environments with inseparable resources, and, moreover, their execution time grows substantially with increase of the slot number. Assuming that every node has at least one local job scheduled, the backfill algorithm has quadratic complexity in the slot number. Although backfilling supports multiprocessor jobs and is able to find a rectangular window of concurrent slots, this can be done provided that all available computational nodes have equal performance (processor clock speed),

and tasks of any job are homogeneous. We take a step further, so proposed algorithms deal with heterogeneous resources and jobs, and can form non-rectangular time-slot windows as a result.

This work is organized as follows. Section 2 introduces a main scheduling scheme. In section 3 two algorithms for search of alternative slot sets are considered. The example of slot search is presented in section 4. Simulation results for comparison of proposed algorithms are described in Section 5. Section 6 summarizes the work and describes further research topics.

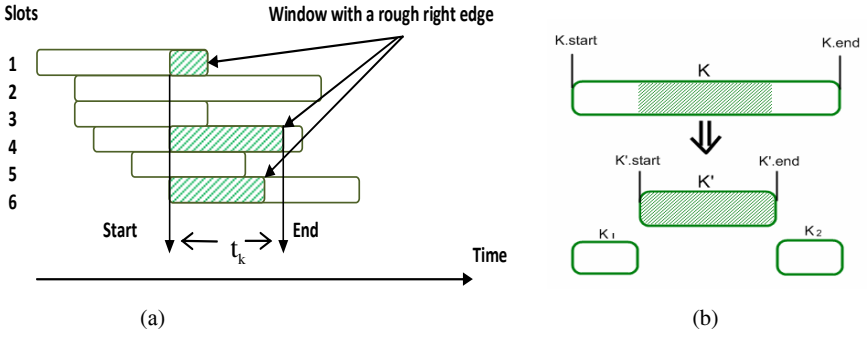
2 Main Scheduling Scheme

The job scheduling is finding a set of time slots. The resource requirements are arranged into a resource request containing the usage time t and the characteristics of computational nodes (clock speed, RAM volume, disk space, operating system etc.).

Let $J = \{j_1,...,j_n\}$ denote the batch consisting of n jobs. A slot set fits a job $j_i, i = 1,...,n$, if it meets the requirements of number and type of resources, cost c_i and the job execution time t_i. We suppose that for each job j_i in the current scheduling cycle there is at least one suitable set s_i. Otherwise, the scheduling of the job is postponed to the next iteration. Every slot set s_i for the execution of the i-th job in a batch $J = \{j_1,...,j_n\}$ is defined with a pair of parameters, the cost $c_i(s_i)$ and the time $t_i(s_i)$ of the resource usage, $c_i(s_i)$ denotes a total cost of slots in a set and $t_i(s_i)$ denotes the execution time of the i-th job.

During every cycle of the job batch scheduling two problems have to be solved.

1. Selecting alternative sets of slots (alternatives) that meet the requirements (resource, time, and cost).
2. Choosing a slot combination that would be the efficient or optimal in terms of the whole job batch execution in the current cycle of scheduling.



(a) (b)

Fig. 1 Slot selection: an ordered list of available slots (a); slot subtraction (b)

To realize the scheduling scheme described above, first of all, we need to propose the algorithm of finding a set of alternative slot sets**.** Slots are arranged by start time in non-decreasing order (Fig. 1 (a)). In the case of homogeneous nodes, a set of slots for any job is represented with a rectangle window. In the case of CPUs with varying performance, that will be a window with a rough right edge, and the usage time is defined by the execution time t_k of the job part (task) that is using the slowest CPU.

This scheme works iteratively, during the iteration it consequentially searches for a single alternative for each job of the batch. In case of successful slot selection for the i-th job, the list of viewed slots for the $(i+1)$-th job is modified. All time spans that are involved in the i-th job alternative are excluded from the list of vacant slots (Fig. 1 (b)). The selection of slots for the $(i+1)$-th job is performed on the list modified with the method described above. Suppose, for example, that there is a slot K' among the appropriate window slots. Then its start time equals to the start time of the window: K′.startTime = window.startTime and its end time equals to K′.end=K′.start $+ t_k$, where t_k is the evaluation of a task runtime on the CPU node, on which the slot is allocated. Slot K' should be subtracted from the original list of available system slots. First, we need to find slot K – the slot, part of which is K' and then cut K' interval from K. So, in general, we need to remove slot K' from the ordered slot list and insert two new slots K_1 and K_2. Their start, end times are defined as follows: K_1.startTime = K.startTime, K_1.endTime = K′.startTime, K_2.startTime = K′.endTime, K_2.endTime = K.endTime. Slots K_1 and K_2 have to be added to the slot list given that the list is sorted by non-decreasing start time order (see Fig. 1 (a)). Slot K_1 will have the same position in the list as slot K, since they have the same start time. If slots K_1 and K_2 have a zero time span, it is not necessary to add them to the list.

After the last of the jobs is processed, the algorithm starts next iteration from the beginning of the batch and attempts to find other alternatives on the modified slot list. Alternatives found do not intersect in processor time, so every job could be assigned to some set of found slots without the revision of other jobs assignments. The search for alternatives ends when on the current list of slots the algorithm cannot find any suitable set of slots for any of the batch jobs.

An optimization technique for the second phase of this scheduling scheme was proposed in [11]. It is implemented by dynamic programming methods using multiple criteria in accordance with the VO economic policy.

We consider two types of criteria in the context of our model. These are the execution cost and time measures for the job batch J using the suitable slot set $\bar{s} = (s_1,...,s_n)$. The first criteria group includes the total cost of the job batch execution $C(\bar{s}) = \sum_{i=1}^{n} c_i(s_i)$. The VO administration policy and, partially, users' interests are represented with the execution time criterion for all jobs of the batch

$T(\bar{s}) = \sum_{i=1}^{n} t_i(s_i)$. In order to forbid the monopolization of some resource usage by users, a limit B^* is put on the budget of the VO that is the maximum value for a total usage cost of resources in the current scheduling cycle. The total slots occupancy time T^* represents owners' urge towards the balance of global (external) and local (internal) job shares.

Let $g_i(s_i)$ be the particular function, which determines the efficiency of s_i slot set usage for i-th job. In other words, $g_i(s_i) = c_i(s_i)$ or $g_i(s_i) = t_i(s_i)$. Let $f_i(Z_i)$ be the extreme value of the particular criterion using a slot set s_i for execution of jobs $i, i+1,..., n$, having Z_i as a total occupancy time or the usage cost of slots $s_i, s_{i+1},..., s_n$ for jobs $j_i, j_{i+1},..., j_n$. Let us define an admissible time value or a slot occupancy cost as $z_i(s_i)$. Then $z_i(s_i) \le Z_i \le Z^*$, where Z^* is the given limit. For example, if $z_i(s_i) = t_i(s_i)$, then $t_i(s_i) \le T_i \le T^*$, where T_i is a total slots occupancy time for jobs $i, i+1,..., n$ and T^* is the constraint for values T_i, that is chosen with the consideration of balance between the global job flow (user-defined) and the local job flow (owner-defined). If, for example, $z_i(s_i) = c_i(s_i)$, then $c_i(s_i) \le C_i \le B^*$, where C_i is a total cost of the resource usage for the tasks $i, i+1,..., n$, and B^* is the budget of the VO. In the scheme of backward run [13], $Z_1 = Z^*$, $i = 1$, $Z_i = Z_{i-1} - z_{i-1}(s_{i-1})$, having $1 < i \le n$.

The functional equation for obtaining a conditional (given $z_i(s_i)$) extremum of $f_i(z_i(s_i))$ for the backward run procedure can be written as follows:

$$f_i(Z_i) = \operatorname*{extr}_{s_i}\{g_i(s_i) + f_{i+1}(Z_i - z_i(s_i))\},\ i = 1,..., n,\ f_{n+1}(Z_{n+1}) \equiv 0. \qquad (1)$$

If we consider the single-criterion optimization of the job batch execution, then every criterion $C(\bar{s})$ or $T(\bar{s})$ must be minimized with given constraints T^* or B^* for the interests of the particular party (a user, an owner and the VO administrator).

For example, a limit put on the total time of slot occupancy by tasks may be expressed as:

$$T^* = \sum_{i=1}^{n} \sum_{s_i} [t_i(s_i)/l_i], \qquad (2)$$

where l_i is the number of admissible slot sets for the i-th job; $[\cdot]$ means the nearest to $t_i(s_i)/l_i$ not greater integer.

The VO budget limit B^* may be obtained as the maximal income for resource owners according to (1) with the given constraint T^* defined by (2):

$$B* = \max_{s_i}\{c_i(s_i) + f_{i+1}(T_i - t_i(s_i))\}. \quad (3)$$

In the general case of the model [7], it is necessary to use a vector of criteria, for example, $<C(\bar{s}), D(\bar{s}), T(\bar{s}), I(\bar{s})>$, where $D(\bar{s}) = B* - C(\bar{s})$, and $I(\bar{s}) = T* - T(\bar{s})$.

3 Slot Search Algorithms

Let us consider one of the resource requests associated with a job in a batch J. The resource requests specifies following: *N* concurrent time-slots providing resource performance rate at least P and maximal resource price not higher, than C, should be reserved for time span t. Here a slot search algorithm for a single job and resource charge per time unit is described.

It is an **A**lgorithm based on **L**ocal **P**rice of slots (ALP) with *a restriction to the cost of individual slots*. Source data include available slots list, and slots being sorted by start time in ascending order (see Fig. 1(a)). The search algorithm requires a sorted list to function and guarantees examination of every slot if this requirement is fulfilled.

1°. Sort the slots by start time in ascending order - see Fig. 1 (a).

2°. From the resulting slot list the next suited slot s_k is extracted and examined.

Slot s_k suits, if following conditions are met:

a) resource performance rate $P(s_k) \geq P$ for slot s_k;

b) slot length (time span) is enough (depending on the actual performance of the slot's resource) $L(s_k) \geq t * P(s_k)/P$;

c) resource charge per time-unit $C(s_k) \leq C$.

If conditions **a)**, **b)**, and **c)** are met, the slot s_k is successfully added to the window list.

3°. We add a time offset d_k of current k-th slot in relation to $(k-1)$-th to the length of the window.

4°. Slots whose length has expired considering the offset d_k are removed from the list. The expiration means that remaining slot length $L'(s_k)$, calculated like shown in **step 2°b**, is not enough assuming the k-th slot start is equal to the last added slot start: $L'(s_k) < (t - (T_{last} - T(s_k)))P(s_k)/P$, where $T(s_k)$ is the slot's start time, T_{last} is the last added slot's start time.

5°. Go to **step 2**°, until the window has N slots.

6°. **End** of the algorithm.

We can move only forward through the slot list. If we run out of slots before having accumulated N slots, this means a failure to find the window for a job and its scheduling is postponed by the metascheduler until the next scheduling cycle. Otherwise, the window becomes an alternative slot set for the job. ALP is executed cyclically for every job in the batch $J = \{j_1,..., j_n\}$. Having succeeded in the search for window for the j_i-th job, the slot list is modified with subtraction of formed window slots (see Fig. 1 (b)). Therefore slots of the already formed slot set are not considered in processing the next job in the batch.

In the economic model [11] a user's resource request contains the maximal resource price requirement, that is a price which a user agrees to pay for resource usage. But this approach narrows the search space and restrains the algorithm from construction of a window with more expensive slots. The difference of the next proposed algorithm is that we replace maximal price C requirement by *a maximal budget of a job*.

It is an **A**lgorithm based on **M**aximal job **P**rice (AMP). The maximal budget is counted as $S = CtN$, where t is a time span to reserve and N is the necessary slot number. Then, as opposed to ALP, the search target is a window, formed by slots, whose total cost will not exceed the maximal budget S. In all other respects, AMP utilizes the same source data as ALP.

Let us denote additional variables as follows: N_S – current number of slots in the window; M_N – total cost of first N slots.

Here we describe AMP approach for a single job.

1°. Find the earliest start window, formed by N slots, using ALP excluding the condition **2°c** (see ALP description above).

2°. Sort window slots by their cost in ascending order.

Calculate total cost of first N slots M_N.

If $M_N \le S$, go to **4**°, so the resulting window is formed by first N slots of the current window, others are returned to the source slot list. Otherwise, go to **3**°.

3°. Add the next suited slot to the list following to conditions **2°a** and **2°b** of ALP. Assign the new window start time and check expiration like in the **step 4**° of ALP.

If we have $N_S < N$, then repeat the current step. If $N_S \ge N$, then go to **step 2**°.

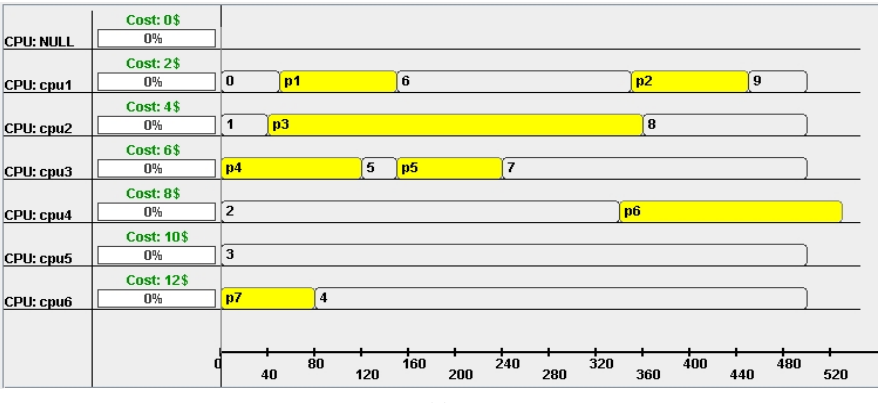
If we ran out of slots in the list, and $N_S < N$, then we have algorithm failure and no window is found for the job.

4°. **End** of the algorithm.

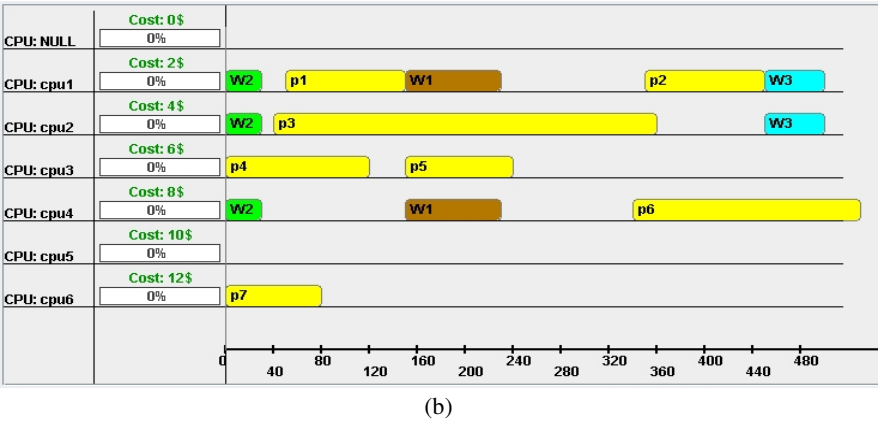
We can state three main features that distinguish the proposed algorithms. First, both *algorithms* consider resource performance rates. This allows forming time-slot windows with *uneven* right edge (we suppose that all concurrent slots for the job must start simultaneously). Second, both algorithms consider maximum price constraint which is imposed by a user. Third, both algorithms have linear complexity $O(m)$, where m is the number of available time-slots.

4 AMP Search Example

In this example for the simplicity and ease of demonstration we consider the problem with a uniform set of resources, so the windows will have a rectangular shape without the rough right edge. Let us consider the following initial state of the distributed computing environment. In this case there are six processor nodes cpu1-cpu6 (Fig. 2 (a)). Each has its own unit cost (cost of it's usage per time unit), which is listed in the column to the right of the processor name. In addition there are seven local tasks p1-p7 already scheduled for the execution in the system under consideration. Available system slots are drawn as rectangles 0...9 - see Fig. 2 (a). Slots are sorted by non-decreasing time of start and the order number of each slot is indicated on its body. For the clarity, we consider the situation where the scheduling cycle includes the batch of only three jobs with the following resource requirements.



(a)



(b)

Fig. 2 AMP: initial state of environment (a); alternatives found after the first iteration (b)

Job 1 requirements: the number of required processor nodes: 2; runtime: 80; maximum total "window" cost per time: 10.

Job 2 requirements: the number of required processor nodes: 3; runtime: 30; maximum total "window" cost per time: 30.

Job 3 requirements: the number of required processor nodes: 2; runtime: 50; maximum total "window" cost per time: 6.

According to AMP alternatives search, first of all, we should form a list of available slots and find the earliest alternative (the first suitable window) for the first job of the batch. We assume that **Job 1** has the highest priority, while **Job 3** possesses the lowest priority. The alternative found for **Job 1** (see Fig. 2 (b)) has two rectangles on cpu1 and cpu4 resource lines on a time span [150, 230] and named W1.

The total cost per time of this window is 10. This is the earliest possible window satisfying the job's resource request. Note that other possible windows with earlier start time are not fit the total cost constraint. Then we need to subtract this window from the list of available slots and find the earliest suitable set of slots for the second batch job on the modified list. Further, a similar operation for the third job is performed (see Fig. 2 (b)). Alternative windows found for each job of the batch are named W1, W2, and W3 respectively. The earliest suitable window for the second job (taking into account alternative W1 for the first job) consists of three slots on the cpu1, cpu2 and cpu4 processor lines with a total cost of 14 per time unit. The earliest possible alternative for the third job is W3 window on a time span of [450, 500]. Further, taking into account the previously found alternatives, the algorithm performs the searching of next alternative sets of slots according to the job priority. The algorithm works iteratively and makes an attempt to find alternative windows for each batch job at each iteration. Figure 3 illustrates the final chart of all alternatives found during search.

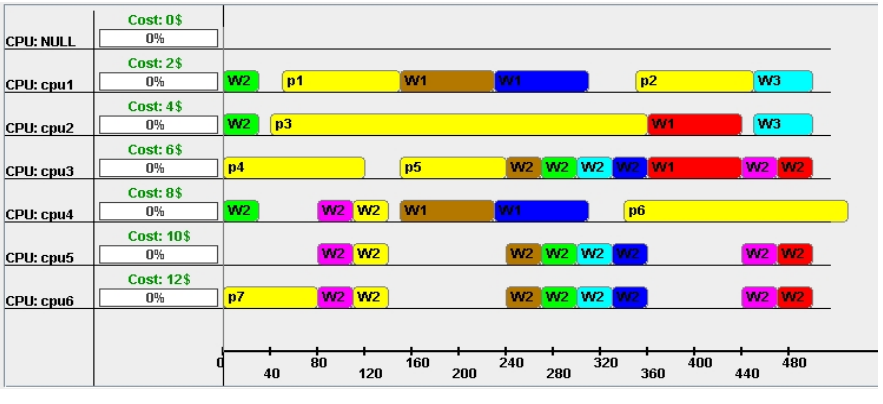


Fig. 3 AMP: the final chart of all alternatives found

Note that in ALP approach the restriction to the cost of individual slots would be equal to 10 for **Job 2** (as it has a restriction of total cost equals to 30 for a window allocated on three processor nodes). So, processor cpu6 with a 12 usage cost

value is not considered during the alternative search with ALP algorithm. However it is clear that in the presented AMP approach eight alternatives have been found. They use the slots allocated on cpu6 line, and thus fit into the limit of the window total cost.

5 Simulation Studies

The experiment consists in comparison of job batch scheduling results using different sets of suitable slots found with described above AMP and ALP approaches. The alternatives search is performed on the same set of available vacant system slots. During the single simulated scheduling cycle the generation of an ordered list of vacant slots and a job batch is performed. To perform a series of experiments we found it more convenient to generate an ordered list of available slots (see Fig. 1 (a)) with preassigned set of features instead of generating the whole distributed system model and obtain available slots from it. **SlotGenerator** and **JobGenerator** classes are used to form the ordered slot list and the job batch during the experiment series. Here is the description of the input parameters and values used during the simulation.

SlotGenerator:

- number of available system slots in the ordered list varies in [120, 150];
- length of the individual slot in [50, 300];
- computational nodes performance range is [1, 3];
- the probability that the nearby slots in the list have the same start time is 0.4;
- the time between neighbor slots in the list is in [0, 10];
- the price of the slot is randomly selected from [0.75p, 1.25p], where p= (1.7) to the (Node Performance).

JobGenerator:

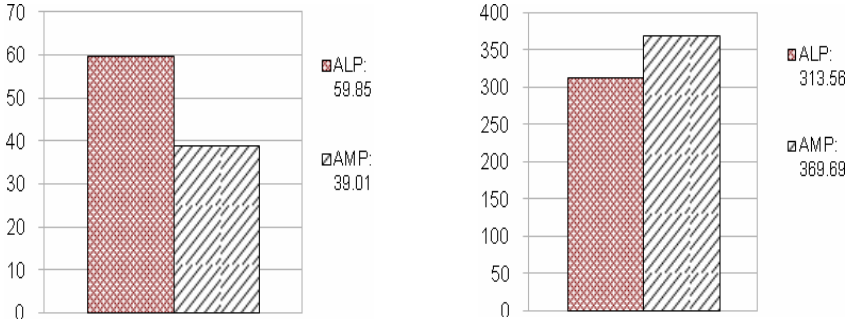
- number of jobs in the batch [3, 7];
- number of computational nodes to find is in [1, 6];
- length (representing the complexity) of the job [50, 150];
- the minimum required nodes performance [1, 2].

All job batch and slot list options are random variables that have a uniform distribution inside the identified intervals.

Let us consider the task of a slot allocation during the ***job batch total execution time minimization*** by the formula (1): $\min_{s_i} T(\bar{s})$ with the constraint B^* in (3). We assume that in (1): $f_i(C_i) = \infty$ given $C_i = 0$.

The number of 25000 simulated scheduling cycles was carried out. Only those experiments were taken into account when all of the batch jobs had at least one suitable alternative of execution. When compared to the target optimization

criterion, AMP algorithm exceeds ALP on 35%. Average batch job total execution time for alternatives found with ALP was 59.85, and for alternatives found with AMP: 39.01 (Fig. 4 (a)). It should be noted, that average cost of batch job execution for ALP method was 313.56, while using AMP algorithm average job execution cost was 369.69, that is 15% more – see Fig. 4 (b).



(a) (b)

Fig. 4 Job batch execution time minimization: job execution time (a); job execution cost (b)

Scheduling results comparison for the first 300 experiments can be viewed in Figure 5. It shows an observable gain of AMP method in every single experiment. The total number of alternatives found with ALP was 258079 or an average of 7.39 for a job. At the same time the modified approach (AMP) found 1160029 alternatives or an average of 34.28 for a single job.

According to the results of the experiment we can conclude that the use of AMP minimizes the total batch execution time though the cost of the execution increases. Relatively large number of alternatives found increases the variety of choosing the efficient slot combination [11] using the AMP algorithm.

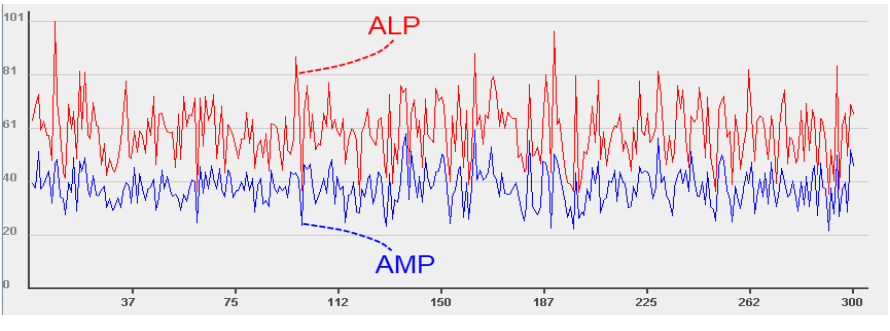
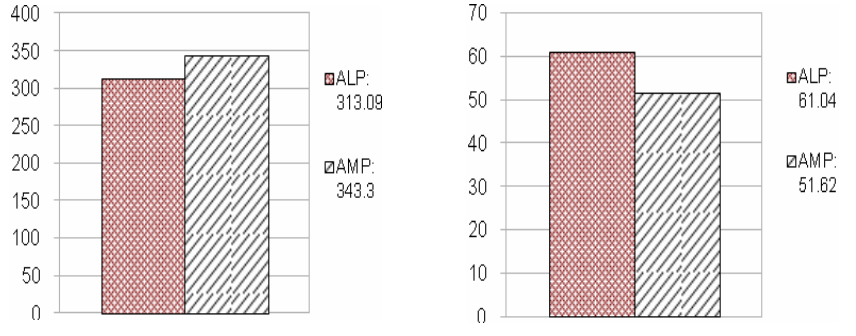


Fig. 5 Execution time comparison for ALP and AMP in job batch execution time minimization

Now let us consider the task of slot allocation during the ***job batch total execution cost minimization*** by the formula (1): $\min_{s_i} C(\bar{s})$ with the constraint T^* in (2). We assume that in (1): $f_i(T_i)=0$ while $T_i=0$.

The results of 8571 single experiments in which all batch jobs were successfully assigned to suitable set of resources using both slot search procedures were collected. Average batch job total execution cost for ALP algorithm was 313.09, and for alternatives found with AMP: 343.3.



(a) (b)

Fig. 6 Job batch total execution cost minimization: job execution cost (a); job execution time (b)

It shows the advantage in the target criterion of only 9% for ALP approach over AMP (Fig. 6 (a)).

Average batch job total execution time for alternatives found with ALP was 61.04. Using AMP algorithm average job execution time was 51.62, that is 15% less than using ALP (Fig. 6 (b)).

Average number of slots processed in a single experiment was 135.11. This number coincides with the average number of slots for all 25000 experiments, which indicates the absence of decisive influence of available slots number to the number of successfully scheduled jobs. Average number of batch jobs in a single scheduling cycle was 4.18. This value is smaller than average over all 25000 experiments. With a large number of jobs in the batch ALP often was not able to find alternative sets of slots for certain jobs and an experiment was not taken into account. Average number of alternatives found with ALP is 253855 or an average of 7.28 per job. AMP algorithm was able to found a number of 115116 alternatives or an average of 34.23 per job. Recall that in previous set of experiments this numbers were 7.39 and 34.28 alternatives respectively.

According to the experimental results it can be argued that AMP allows to find at average more rapid alternatives and to perform jobs in a less time. But the total cost of job execution using AMP is relatively higher.

6 Conclusions and Future Work

In this work, we address the problem of independent batch jobs scheduling in heterogeneous environment with inseparable resources.

The feature of the approach is searching for a number of job alternative executions and consideration of economic policy in VO and financial user requirements on the stage of a single alternative search.

For this purpose ALP and AMP approaches for slot search and co-allocation were proposed and considered. When compared to the target optimization criteria during the total batch execution time minimization AMP exceeds ALP significantly. At the same time during the total execution cost minimization the gain of ALP method is negligible. It is worth noting, that on the same set of vacant slots AMP in comparison with ALP finds several time more execution alternatives. In our further work we will address the problem of slot selection for the whole job batch at once and not for each job consecutively.

The necessity of guaranteed job execution at the required QoS causes taking into account the distributed environment dynamics, namely, changes in the number of jobs for servicing, volumes of computations, possible failures of processor nodes, etc. [10, 14]. As a consequence, in the general case, a set of versions of scheduling, or a strategy, is required instead of a single version [10, 14].

In future we will refine resource co-allocation algorithms in order to integrate them with scalable co-scheduling strategies [14].

Acknowledgements. This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Leading Scientific Schools (SS-7239.2010.9), the Russian Foundation for Basic Research (grant no. 09-01-00095), the Analytical Department Target Program "The higher school scientific potential development" (projects nos. 2.1.2/6718 and 2.1.2/13283), and by the Federal Target Program "Research and scientific-pedagogical cadres of innovative Russia" (State contracts nos. P2227 and 16.740.11.0038).

References

[1] Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility Grids: time and cost trade-off management. In: Proc. of ACSC 2009, pp. 151–159. Wellington, New Zealand (2009)

[2] Ailamaki, A., Dash, D., Kantere, V.: Economic aspects of cloud computing. Flash Informatique, Special HPC, 45–47 (October 27, 2009)

[3] Bredin, J., Kotz, D., Rus, D.: Economic markets as a means of open mobile-agent systems. In: Proc. of MAC 3, Seattle, USA, pp. 43–49 (1999)

[4] Kurowski, K., Nabrzyski, J., Oleksiak, A., et al.: Multicriteria aspects of Grid resource management. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management. State of the art and future trends. Kluwer Academic Publishers, Boston (2003)

[5] Toporkov, V.: Application-level and job-flow scheduling: An approach for achieving quality of service in distributed computing. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 350–359. Springer, Heidelberg (2009)

[6] Toporkov,V.V.: Job and application-level scheduling in distributed computing. Ubiquitous Comput. Commun. J. 4, 559–570 (2009)
[7] Buyya, R., Abramson, D., Giddy, J.: Economic models for resource management and scheduling in grid computing. J. of Concurrency and Computation: Practice and Experience 5(14), 1507–1542 (2002)
[8] Ernemann, C., Hamscher, V., Yahyapour, R.: Economic scheduling in grid computing. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 128–152. Springer, Heidelberg (2002)
[9] Toporkov, V.V.: Decomposition schemes for synthesis of scheduling strategies in scalable systems. J. Comput. Syst. Sci. Int. 45, 77–88 (2006)
[10] Toporkov, V.V., Tselishchev, A.S.: Safety scheduling strategies in distributed computing. Int. J. Critical Computer-Based Syst. 1-3, 41–58 (2010)
[11] Toporkova, V.V., Toporkova, A., Tselishchev, A., Yemelyanov, D., Bobchenkov, A.: Economic models of scheduling in distributed systems. In: Walkowiak, T., Mazurkiewicz, J., Sugier, J., Zamojski, W. (eds.) Monographs of System Dependability. Dependability of Networks. Oficyna Wydawnicza Politechnki Wroclawskiej, Wroclaw (2010)
[12] Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. on Parallel and Distributed Systems 6(12), 529–543 (2001)
[13] Taha, H.: Operations research: an introduction. Macmillan, New York (1982)
[14] Toporkova, V.V., Toporkova, A., Tselishchev, A., Yemelyanov, D.: Scalable co-scheduling strategies in distributed computing. In: Proc. of the 2010 ACS/IEEE Int. Conf. on Computer Systems and Applications. IEEE CS Press, Los Alamitos (2010)

Functional Based Reliability Analysis of Web Based Information Systems

Tomasz Walkowiak[1] and Katarzyna Michalska[2]

[1] Institute of Computer Engineering, Control and Robotics, Wroclaw University of Technology, ul. Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
e-mail: Tomasz.Walkowiak@pwr.wroc.pl
[2] Institute of Computer Engineering, Control and Robotics, Wroclaw University of Technology, ul. Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
e-mail: Katarzyna.Michalska@pwr.wroc.pl

Abstract. The chapter presents a method of Web-based information system dependability analysis based on functional and reliability approach. A service availability is predicted by developed by authors a two level simulator (using Monte-Carlo based estimation). Taxonomy of the faults is described. Numerical experiments were performed on a test case scenario that was modeled as a network of interacting tasks and technical infrastructure required for the service realization.

1 Introduction

In a decade of extraordinary rapid development of internet terms Web-based systems or service oriented information systems, are used very often and in many cases in the same content. Some researchers suggest [5], [7] that there are two types of web-based systems: Web Sites and Web Application. In [5] this diversification is based on a complexity and purpose of the Internet systems, since first group is treated as a traditional information sites, where the other is seen as a more complex systems (really full-blown information systems, with complex database interactions, using the Internet as a user interface [5]). However no matter the classification and its purpose still common property of all these services is growing threat connected with unexpected and aleatory events (software or hardware failures) or intentional un-friendly activities (attacks) [1]. These problems are even more stressed by internet providers, as a need of a scalable and inexpensive technology that will help to increase safety [1], reliability [1] and dependability [1] of their systems. Type and number of faults covered by the system is playing a primary role in determining the dependability level that the system provides [4]. Considering the variety and multiplicity of fault types, calculation of system dependability become a challenging task, that should be supported with a complex model and powerful tools.

This chapter starts with a presentation of taxonomy of faults. Next, it presents a formal model (section 3) of a Web-based system that can be used for further dependability analysis. Proposed solution is based on the simulation – based on

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 257–269.
springerlink.com

open source simulation framework (section 4.1). The analysis model allows to performance calculate metrics (described in next sections of 4) that allows to analyze dependability aspect of an exemplary case study (described in section 5). The chapter is summarized in section 6.

2 Taxonomy of Faults

There are numerous sources of faults in a complex Web system. These encompass hardware malfunctions (transient and persistent), software bugs, human mistakes, exploitation of software vulnerabilities, malware proliferation, drainage type attacks on system and its infrastructure (such as ping flooding, DDOS). We propose a classification of the faults that is not based on its primary source, but on the effect it has on the system.

Since our research is strongly based on a Web services we relate the effects to a possible cause. Since different faults imply different reactions, that is why no common pattern can be used, but some of the similarities can be specified. Research on this field were done in various works, but since we focus mainly on Web services we based our observation on article [2], that provide a classification of the faults as a three major groups: physical, development and interaction faults.

Physical faults are those that can be detect as a failure in the network infrastructure. Infrastructure of the network i.e. links, host, network medium are one of the crucial points of service unavailability, since when one link or one web server is down, than whole service is down. The only why to fix the physical fault is to replace the element or reconfigure the service, so it would work on a different element.

Next group of faults are the *development faults* mainly caused by human developers, development tools and production facilities [4].

Third group are an *interaction faults* understood as a two categories [4]: content faults (incorrect service, misunderstood behaviour, response error, Quality of Service faults, Service Level Agreement faults) and timing faults (time outs, workflow faults).

Particularly in this chapter, we focus mainly on the physical faults and interaction faults that affect service response.

3 Web Based Information System Model

3.1 Information System Functional Model

Information systems from reliability and functional point of view are complex technical systems [5]. Their complexity and a set of relationships between applications as much links and communication aspect makes analysis of the system a challenging task. The correct system analysis requires to model a system as closely reality as possible. However by increasing the system details one risks uselessness of such analysis due to computational complexity [12]. Considering this problem, we propose to model an information system from a business point of

view – that can be also named service point of view. This abstract view give as an opportunity to calculate some dependability metrics for the system that will be focused on service and user requirements.

Clients expect from the system that it will provide some service in infrastructure that is located on a provider side and with a suitable configuration. User expects to receive a solution for the task that was send to the system as a request. Therefore, we can model IS as a 4-tuple [12]:

$$IS.BS = \langle Client, BSer, TInf, Conf \rangle \tag{1}$$

Client – finite set of clients,
BSer – business service, a finite set of service components,
TInf – technical infrastructure,
Conf – information system configuration.

The client (*Client*) consists of set of users where each user is defined by its allocation (host), number of concurrently ruing users of a given type, set of activities (a sequence of task calls that are realized between service components) and inter-activity delay time.

Business service (*BSer*) is as a kind of business-level abstraction that implements a meaningful business process or business task, and interfaces with users or other business services [8]. In other words, it is a set of services based on business logic that can be used for handling a business process. *BS* is modeled a set of business service components which consists of a set of tasks that are the lowest level entities [12]. It can be seen as a request and response from one system component to another. Each task is described by its name and task processing time parameter (parameter that describes computational complexity the task).

Technical infrastructure (*TInf*) is considered to be a set of devices (host, routers, etc.) and computer network. Network infrastructure is considered as a collection of individual networks and inter-network. We have assumed negligible aspects of TCP/IP traffic [12]. Therefore, we model the network communication as a random delay. Service-based system is containing mostly servers. Each of it is described by a unique name, host performance parameter – the real value which is a base for calculating the task processing time (described later) and a set of technical services (i.e. Tomcat, Oracle database, etc.), where as each technical service is described by a name and a limit of tasks concurrently being executed.

System configuration (*Conf*) is a function that gives the assignments of each service components to a technical service that are placed on a given hosts.

More formal description a presented model could be found in [8].

3.2 Service Response Time Model

As it was mentioned in the previous section we model the system from the user point of view. Therefore, the model and the simulator should allow calculating the service response time.

Let's look how user requests are being executed in service based information systems. The user initiate the communication requesting some tasks on a host, it could require a request to another host or hosts, after the task execution a host responds to requesting server, and finally the user receives the respond. Requests and responds of each task give a sequence of a user task execution – so called choreography. Assume that the choreography for some user c_i is given as a sequence of requests [12]:

$$choregraphy(c_i) = \left(c(task_{b_1}), c(task_{b_2}), ..., c(task_{b_n})\right), \tag{2}$$

where $c(task_{b_i})$ could be a request ($\Rightarrow$) to $task_{b_i}$ or a response from a given task ($\Leftarrow$).

Some tasks are executed on a given host a request is returned to a sender. However, other tasks could require execution of other tasks. Therefore, the task could be described by a sequence of requests, i.e. list of tasks to be called.

For example, some choreography could be written as:

$$choregraphy(c_1) = c_1 \Rightarrow task_1 \Rightarrow task_2 \Leftarrow task_1 \Rightarrow task_3 \Leftarrow task_1 \Leftarrow c_1. \tag{3}$$

Therefore, a user request processing time is equal to time of communication between hosts on which each tasks from the above choreography is placed and the time of processing each task. Therefore, for a given example of choreography (assuming some allocation of tasks) the user request processing time is equal to:

$$\begin{aligned} urpt(c_1) = {} & delay(h_0, h_1) + pt(task_1) + delay(h_1, h_2) + pt(task_2) + \\ & delay(h_2, h_1) + delay(h_1, h_3) + pt(task_3) + delay(h_3, h_1) + delay(h_1, h_0) \end{aligned}. \tag{4}$$

where $delay(h_i, h_j)$ is the time of transmitting the requests from host h_i to h_j, and $pt(task)$ is the time of processing a requests on given host (on the host of which the task is allocated).

The time of transmitting the requests is modeled a random variable by truncated normal distribution.

A task processing time in Web systems depends on the type of task (its computational complexity), type of host (its computational performance) and number of other tasks being executed in parallel. These numbers is changing in time during system lifetime. Therefore, it is hard to use analytic method to calculate formula (4). That is way simulation approach was proposed.

4 Dependability Analysis

4.1 Overview of the Approach

The presented system model was developed to facilitate the dependability analysis of the information systems. More precisely, it is used to determine if the required

system resources (hosts performance and reliability parameters) for a given service choreography and configuration (understood as allocation of each service component to a technical service that are placed on a given hosts) that ensure proper operation at a required level of probability.

The dependability analysis is performed using a Monte Carlo [6] simulator, custom designed for this purpose at Wroclaw University of Technology. It is based on the publicly available SSF simulation engine that provides all the required simulation primitives and frameworks, as well as a convenient modeling language DML (SSF [10]) for inputting all the system model parameters. By repeating the simulator runs multiple times using the same model parameters, we obtain several independent realizations of the same process (the results differ, since the system model is not deterministic). These are used to build the probabilistic distribution of the results, especially the average measures.

The key extension and feature, during simulation process is the calculation of the task processing time. It has to take into account the consumption of computational resources (mainly host processing power). The calculation of task processing time is based on simulating the time-division among each tasks executed at the same time. A detailed description of this approach is outside the scope of this presentation. It can be found in the papers: [12].

Avizienis, Laprie and Randell introduced the idea of service dependability to provide a uniform approach to analyzing all aspects of providing a reliable service. They described [1] basic set of dependability attributes: availability, reliability, safety, confidentiality, integrity and maintainability. This is a base of defining different dependability metrics used in dependability analysis of computer systems and networks.

As it was mentioned, we focused on functional based metrics. Therefore, we consider dependability of an information system as a property of the system to reliable process user tasks. In other words the tasks have to perform not only without faults but also with demanded performance parameters.

4.2 Availability Metric

The system availability is usually defined as the probability that the system is operational (provides correct responses) at a specific time. It is shown that availability is asymptotically equal to the ratio of total system uptime t_{up} to the operation time t, i.e.

$$A = \lim_{t \to \infty} \frac{t_{up}}{t}. \tag{5}$$

Assuming a uniform rate of requests, the asymptotic assessment of availability may be further transformed to average over simulations:

$$A = E\left[\frac{N_{OK}}{N}\right], \tag{6}$$

where N_{OK} is the number of requests correctly handled by the system exposed to a stream of N requests.

4.3 Demanded Performance Parameters

The definition (7) raises the question what does it mean not correctly handled requests. There could be different reasons of not correctly handled requests in real serviced based information system. We will omit here the hardware and software failures and there results since they will be discussed in the next section. Here we will focus only on functional aspects.

The performance of any information system has a big influence on the business service quality. It has been shown [11] that if user will not receive answer from the system in less than 10 seconds he/she will probably resign from active interaction with the system and will be distracted by other ones. Therefore, all requests answered outside this limit time are recognize as not correctly answered.

Moreover, there are two more sources of not correct handled requests coming from the system itself that are presented in our model of information systems: timeouts and services concurrent task limits. The communication protocols (HTTP) as well as Web services (for example JSP) have built-in timeouts. If any request is not finished within a given time limit (in most cases it could be set by configuration parameters) is assumed to be failed. The other reason of not correctly handled requests is a limit to a number of tasks handled by a technical service (i.e. Tomcat) at the same time. It could be also set by configuration parameters of any technical service. Since most of the user tasks consist of a sequence of requests (refer to section 3), if one from the sequence fails the whole user task is assumed to be not correctly handled.

4.4 Reliability Model

Reliability is mostly understood as the ability of a system to perform its required functions for a specified period of time [2]. It's is mostly defined as a probability that a system will perform its function during a given period of time. For stationary systems one could calculate stationary reliability as the asymptotic value of reliability. The typical method for reliability analysis is to define system operational states. Next, calculate the probability of the system being in a given state, assess the reliability states as operational or failed and calculate the reliability as expected value of the system being operational. The main problem to use such approach for Web-based systems is to assign some of operational states to operational or fail status. Assume, that we have a system with load balancers [13] and one of load balancing service is not operating, the whole system will still be in operating system; however its performance will drop. To overcome such problems the availability defined by (4) is the most commonly used reliability measure of Web based systems, which could be calculated using proposed here simulation approach.

The previous section introduced not correctly handled requests as a result of system functionality, i.e. result of time-outs and maximum number of requests. We propose to extend it failures to represents information system faults which occur in a random way. Of course, there are numerous sources of faults in complex information systems as presented in section 2. We propose to model all of them from the

point of view of a resulting failure. We assume that system failures could be modeled as a set of two kinds of failures: element failure and downgrade failure.

Each element failure is assigned to one of elements of technical infrastructure: a host, a technical service or a network connection. If a given element is failed tasks allocated on a given element (host or technical service) are failed. As a result user requests that realize a chorography that includes a given "failed" task are not being correctly answered. In case of connection failures the algorithm is similar. If a connection between two hosts is failed the choreography that requires this connection is not correctly handled.

Each element failure is modeled as an independent working-failure stochastic process, i.e.:

$$elelemntfailure =< e, \mu, \lambda >, \tag{7}$$

where

- e – is the system element on which the failure will happen, there could be several failures assigned to one element;
- μ – is mean value of truncated normal distribution (with standard deviation equal to 0.1 of the mean value) which models the repair time, i.e. the time after which the failure will be repaired and the element will come back to normal operation;
- λ - is the intensity of exponential distribution, which models the time between failures.

The downgrade failure models the situation when the host can operate, but it cannot provide the full computational resources, causing some services to fail or increasing their response time above the acceptable limits. This kind of failures could cause the failure of some task requests due to timeout parameter introduced in the previous section.

In real systems the downgrade failures are caused by the undirected attacks such as virus or worms proliferation or directed attacks that usually are based on draining their limited resources, e. g. denial-of-service attacks.

Each downgrade failure is assigned to a given host (h) and similarly to element failure is modeled as an independent working-failure stochastic process (described by μ and λ as it was mentioned above):

$$downgradefailure =< h, pd, \mu, \lambda >, \tag{8}$$

where

- pd – is a numerical value from (0,1) range; it downgrades the host performance, so enlarge the task processing time.

4.5 Two Level Simulations

The reliability model presented above could be a part of simulation algorithm mentioned in section 4.1. Failures could be added to a functional simulator. They are

just a new kind of events occurring in a probabilistic way. A usage of Monte-Carlo approach (i.e. a large number of simulation repetition) the final metric (for example availability from equation) could be calculated. We have presented such approach in [9]. However, it has one big drawback a huge computation complexity. The modeled failures occurs so rare (several times per year for one system element) compared to a task execution time (a hundredth of second) so to achieve numerical stable results of Monte-Carlo based estimation of availability metric one has to simulate almost a trillion (10e+12) user requests (for 100 requests per second).

We propose to overcome this problem by introducing a two level simulation: reliability and functional one. The approach is based on the methodology of system reliability analysis by Markov processes [2], which encompass following stages:

- set of reliability states definition;
- states-transition matrix definition;
- proper system of linear equations construction.

Markov based methodology supports satisfactory results if the system isn't very complicated and if one uses exponential distribution for the state transition times. Both requirements are not fulfilled in the analysed case. Therefore, we have had to modify it.

First, we define the system operational states. For, example for failures of two elements it will give 4 states (as presented in Fig. 1.): S_0 – both element working, S_1 the first failed, S_2 – second failed, S_2 – both failed.

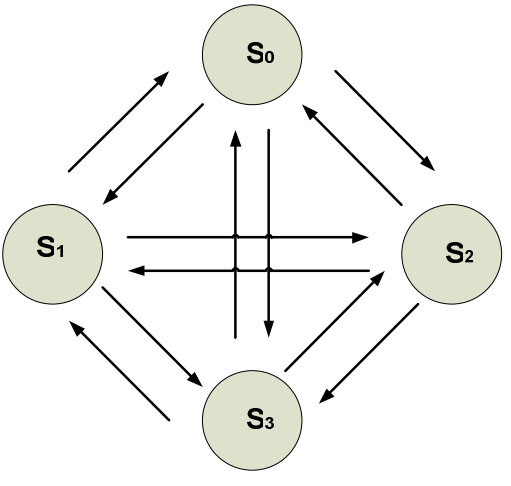


Fig. 1 The reliability states for a system with two failures

Next, we calculate the probability of a system being in a given state ($P(S_i)$) for a stationary state. We use a simulation tool since the normal distribution used for modeling the repair time is not allowed in Markov approach. A SSF tool [10] was also used to develop this reliability level simulator.

For each operational state the availability parameter ($A(S_i)$) is estimated using simulator tool mentioned in section 4.1. The simulation is done with existence of failure or failures in a system.

Finally, the availability could be calculated as an average value of availabilities for each operational state:

$$A = \sum_{i=1}^{N} A(S_i) \cdot P(S_i) \,. \tag{9}$$

The main problem is a number of states. For *n*-independent elements, which could be failed, it gives 2^n states. So for the system described in the next section, that includes 6 hosts and we modeled 6 element failures and 6 downgrade failures it gives 4096 states. Simulations of system in each state are very time consuming tasks. Therefore, we propose to skip the least probable states. It could be done by sorting all states by its probability starting with the most probable, and then by selecting *N* - states that represent more than 99.99% the time:

$$0.9999 < \sum_{i=1}^{N} P(S_i) \,. \tag{10}$$

5 Experiments and Results

5.1 Testbed

For the case study analysis we propose an exemplar service system illustrated in Fig. 2. The system is composed of three networks: one is a client network, other service provider networks (secured by a Firewall). For service realization system contains of a few servers i.e.: *WebServer*, *Flightdatabase, ReservationServer, PaymentServerController, BackupWebServer*.

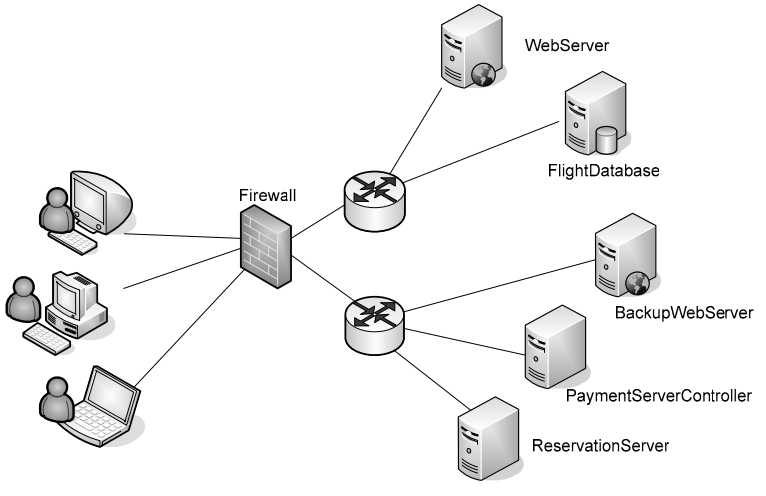


Fig. 2 Testbed system

System realized service connected with booking flight ticket (Fig. 3.). First of all user can select departure. In results flight collection is returned. If user want to make a reservation for a flight he/she chose, payment order is filled and a flight collection is updated (availability of the flight table is changed).

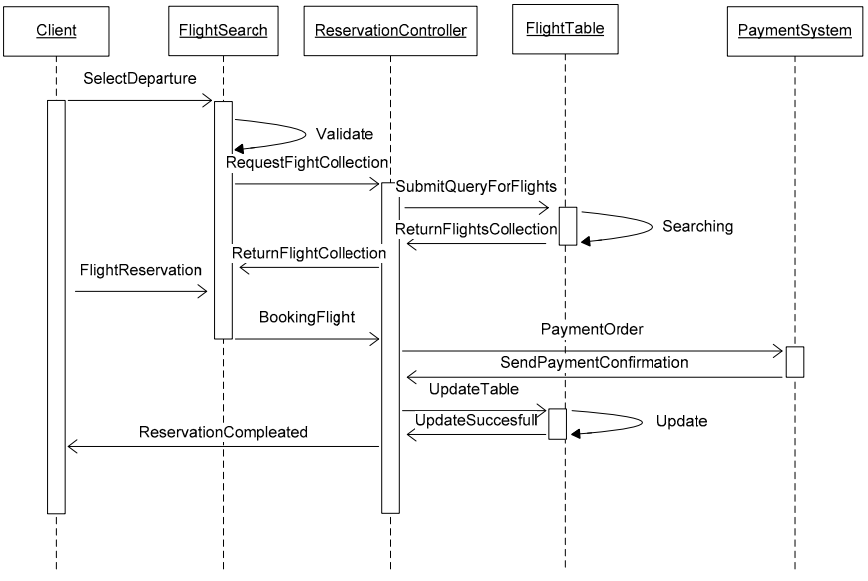


Fig. 3 UML diagram of the testbed system

5.2 Reliability Parameters

The timeouts for all technical services where set to 10 s, whereas a limit to a number of tasks was set to 200. In simulation experiments performed on presented testbed we consider two types of failures for each host: element failure and downgrade failure. The first one represents the results of host or system operation failure. Since, today's computer devices to not failing very often, the intensity was set to one year per year. The second types of faults (with 0.98 downgrade parameter) are modeling any virus or malware occurrence. They are more probable than a host failure, especially for systems that are exposed to attacks. Web-based systems are definitely in this group. Therefore, in our study mean intensity of virus occurrence is set to 2 per one year. The mean time of host fault repair is equal to 6 hours whereas for viruses 3 hours. Threats like viruses occupy large number of a central processor unit (CPU). Therefore, in case of a virus occurrence only 2% of host CPU is available for user requests executions.

5.3 Results

The reliability simulation uses 51 states among 4096 available for a testbed system w (6 element failures: 5 servers and a firewall; and 6 downgrade failures).

The achieved results, the availability for different number of clients per second is presented in Fig. 4. It could be noticed that the system has 0.99 availability for input load equal to 10 and less users per second, for larger number it drops approaching less than 0.01 for more than 1000 users per second.

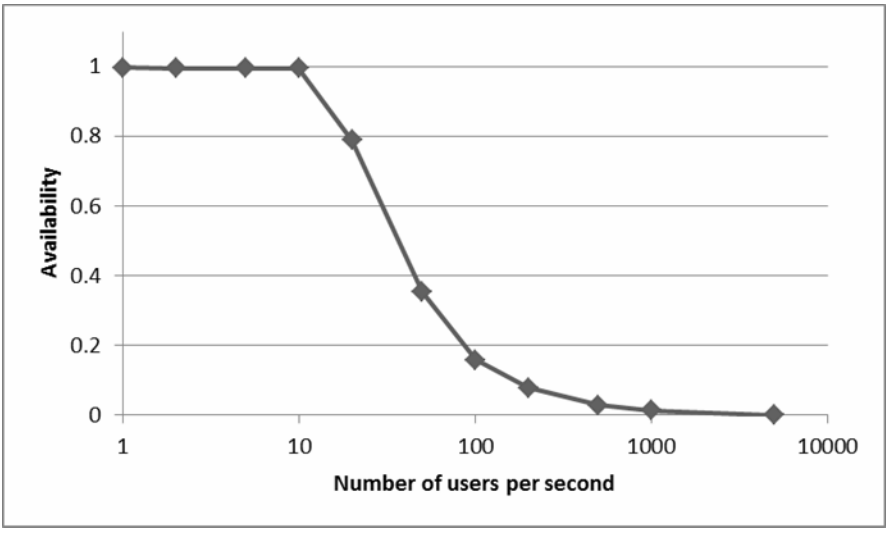


Fig. 4 Availability for a testbed system in a function of number of users per second

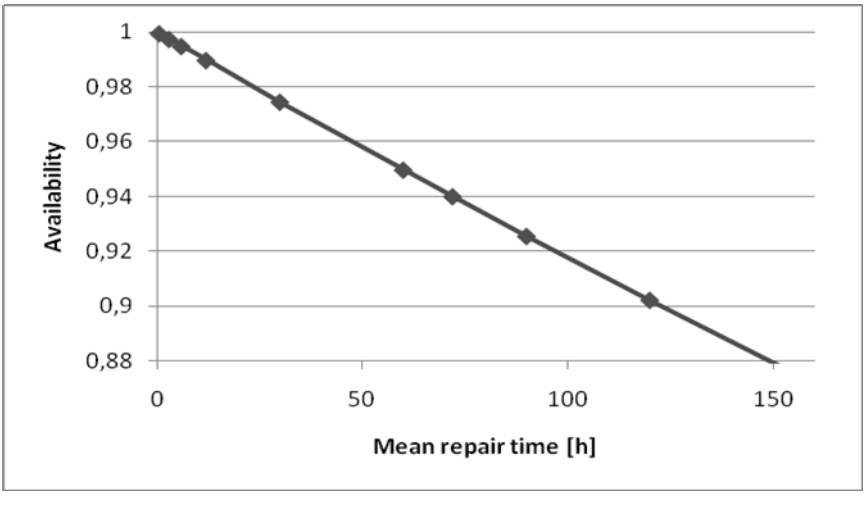


Fig. 5 Availability for a testbed system with 10 users per second in a function of mean repair time

The Fig. 5 presents the availability for a testbed system with 10 users per second in a function of mean repair time. The downgrade failure mean repair time was equal to a half of the mean repair time for hosts (element failure) following idea presented in section 5.2. The results could be a base for setting an agreement with external service company for response time to achieve needed level of availability. The mean repair time from our model is equal to a response time plus time needed for a repair.

6 Conclusion

In this chapter we have presented a functional and reliability analysis of Web-based information systems based on modelling and analysis approach [3].

Developed simulation software allows to analyze the availability (understood as the ability of a system to perform its required functions for a specified period of time) of a given configuration of Web-based information system in a function of functional (like users number) and reliability (like mean repair time) parameters. Using the tool Web-based system can be easily verified, what makes the solution a powerful tool for increasing system dependability and by that increasing user satisfaction. Considering complexity of the Web-based information system, we keep in mind that the model should cover more and more realistic parameters, to improve the analysis even more precisely. Researches in this area are still in prepress, based on workload model and extended faults models.

In the future, we plan to extend our solution with a set of new dependability metrics, based on work that is currently ongoing. Next, we would like to analyze system with a load balancing model to incorporate presented here fault model. It will allow analyzing large Web-based information systems.

Acknowledgment. The presented work was funded by the Polish National Science Centre under contract no. 4759/B/TO2/2011/40.

References

[1] Avizienis, A., Laprie, J., Randell, B.: Fundamental Concepts of Dependability. In: 3rd Information Survivability Workshop, pp. 7–12 (2000)

[2] Barlow, R., Proschan, F.: Mathematical Theory of Reliability. Society for Industrial and Applied Mathematics, Philadelphia (1996)

[3] Birta, L., Arbez, G.: Modelling and Simulation: Exploring Dynamic System Behaviour. Springer, London (2007)

[4] Chan, K.S., Bishop, J., Steyn, J., Baresi, L., Guinea, S.: A Fault Taxonomy for Web Service Composition. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 363–375. Springer, Heidelberg (2009)

[5] Conallen, J.: Building Web Applications with UML. Addison Wesley Longman Publishing Co, Amsterdam (2000)

[6] Fishman, G.: Monte Carlo: Concepts, Algorithms, and Applications. Springer, Heidelberg (1996)

[7] Gold, N., Knight, C., Mohan, A., Munro, M.: Understanding service-oriented software. IEEE Software 21, 71–77 (2004)
[8] Michalska, K., Walkowiak, T.: Simulation approach to performance analysis information systems with load balancer. Information systems architecture and technology: advances in Web-Age Information Systems, Oficyna Wydawnicza Politechniki Wroclawskiej, 269–278 (2009)
[9] Michalska, K., Walkowiak, T.: Fault modelling in service-based oriented information systems. Information systems architecture and technology: new developments in Web-Age Information, Oficyna Wydawnicza Politechniki Wroclawskiej, 89–99 (2010)
[10] Nicol, D., Liu, J., Liljenstam, M., Guanhua, Y.: Simulation of large scale networks using SSF. In: Proceedings of the 2003 Winter Simulation Conference, vol. 1, pp. 650–657 (2003)
[11] Nielsen, J.: Usability Engineering. Published by Morgan Kaufmann, San Francisco (1994)
[12] Walkowiak, T.: Information systems performance analysis using task-level simulator. In: DepCoS - RELCOMEX 2009, pp. 218–225. IEEE Computer Society Press, Los Alamitos (2009)
[13] Walkowiak, T., Michalska, K.: Performance analysis of service-based information system with load balancer - simulation approach. Dependability of networks, Oficyna Wydawnicza Politechniki Wroclawskiej, 155–168 (2010)

Human Resource Influence on Dependability of Discrete Transportation Systems

Tomasz Walkowiak[1] and Jacek Mazurkiewicz[2]

[1] Institute of Computer Engineering, Control and Robotics,
Wroclaw University of Technology, ul. Wybrzeze Wyspianskiego 27,
50-370 Wroclaw, Poland
e-mail: Tomasz.Walkowiak@pwr.wroc.pl
[2] Institute of Computer Engineering, Control and Robotics,
Wroclaw University of Technology, ul. Wybrzeze Wyspianskiego 27,
50-370 Wroclaw, Poland
e-mail: Jacek.Mazurkiewicz@pwr.wroc.pl

Abstract. The chapter is focused on the human resource influence on dependability of discrete transportation systems (*DTS*). The human resource means the driver of the vehicle. We add him/her as a new element of the system description. The dependability means the combination of the reliability and functional parameters of the *DTS*. This way the analysis of the *DTS* behavior seems to be more sophisticated. The unified containers transported by trucks with the set of time-type assumptions are the essence of the system discussed. The proposed method is based on modeling and simulating of the system behavior. The income of containers is modeled by a stochastic process. Each container has a source and destination address. The central node is the destination address for all containers generated in the ordinary nodes. We also propose the heuristic management approach as well as the functional metric for *DTS* and we test the example system based on the real data.

1 Introduction

The transportation systems are characterized by a very complex structure. The performance of the system can be impaired by various types of faults related to the transport vehicles, communication infrastructure or even by traffic congestion or human resource [1]. It is hard for an administrator or an owner to understand the system behaviour. To overcome this problem we propose a functional approach. The transport system is analysed from the functional point of view, focusing on business service realized by a system [16]. The analysis is following a classical [4]: modelling and simulation approach. It allows to calculate different system measures which could be a base for decisions related to administration of the transport systems. The metric are calculated using Monte Carlo techniques [7]. No restriction on the system structure and on a kind of distribution is the main advantage of the method. Such approach allows to forget about the classical reliability analysis based on Markov or Semi-Markov processes [2] - idealised and hard for

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 271–283.
springerlink.com

reconciliation with practice. The chapter is based on the transportation system of the Polish Post regional centre of mail distribution (section 2). The developed discrete transport system model is presented in section 3. The main service given by the post system is the delivery of mails. From the client point of view the quality of the system could be measured by the time of transporting the mail from the source to the destination. A driver is a new element of the system description. We pointed the set of states to characterise the actual driver position including formal – law–origin aspects: number of hours he or she can work daily for example. We offer the heuristic approach to system management (section 4). In our opinion it seems to be the most adequate to the level of detail discussed in the described work. The quality of the analysed system is measured by the availability defined as an ability to realize the transportation task at a required time (described in section 5). Next (section 6), we give an example of using presented model for the analysis of the Dolny Slask Polish Post regional transportation system.

2 Polish Post Transportation System

The analysed transport system is a simplified case of the Polish Post. The main aim of the system is to delivery of mail. The system consists of a set of nodes placed in different geographical locations. Two kinds of nodes could be distinguished: central nodes (*CN*) and ordinary nodes (*ON*). There are bidirectional routes between nodes. Mails are distributed among ordinary nodes by trucks, whereas between central nodes by trucks, railway or by plain. The mail distribution could be understood by tracing the delivery of some mail from point *A* to point *B*. At first the mail is transported to the nearest to *A* ordinary node. Different mails are collected in ordinary nodes, packed in larger units called containers and then transported by trucks scheduled according to some time-table to the nearest central node. In the central node containers are repacked and delivered to appropriate (according to delivery address of each mail) central node. In the Polish Post there are 14 central nodes and more than 300 ordinary nodes. There are more than one million mails going through one central node within 24 hours. It gives a very large system to be modelled and simulated. Therefore, we have decided to model only a part of the Polish Post transport system – one central node with a set of ordinary nodes. Essential in any system modelling and simulation is to define the level of details of modelled system. Increasing the details causes the simulation becoming useless due to the computational complexity and a large number of required parameter values to be given. On the other hand a high level of modelling could not allow to record required data for system measure calculation. Therefore, a crucial think in the definition of the system level details is to know what kind of measures will be calculated by the simulator. Since the business service given by the post system is the delivery of mails on time, we have decided to measure to calculate the time of transporting mails within the system. Since the number of mails presented in the modelled system is very large and all mails are transported

in larger amounts containers, we have proposed to use containers as the smallest observable element of the system. Therefore, the main observable value calculated by the simulator will be the time of container transporting from the source to the destination node. The income of mails to the system, or rather containers of mails as it was discussed above, is modelled by a stochastic process. Each container has a source and destination address. The central node is the destination address for all containers generated in the ordinary nodes. Where containers addressed to any ordinary nodes are generated in the central node. The generation of containers is described by some random process. In case of central node, there are separate processes for each ordinary node. Whereas, for ordinary nodes there is one process, since commodities are transported from ordinary nodes to the central node or in the opposite direction. The containers are transported by vehicles. Each vehicle has a given capacity – maximum number of containers it can haul. Central node is a base place for all vehicles. They start from the central node and the central node is the destination of their travel. The vehicle hauling a commodity is always fully loaded or taking the last part of the commodity if it is less than its capacity. Vehicles operate according to the time-table. The time-table consists of a set of routes (sequence of nodes starting and ending in the central node, times of leaving each node in the route and the recommended size of a vehicle). The number of used vehicle and the capacity of vehicles does not depend on temporary situation described by number of transportation tasks or by the task amount for example. It means that it is possible to realize the route by completely empty vehicle or the vehicle cannot load the available amount of commodity (the vehicle is too small). Time-table is a fixed element of the system in observable time horizon, but it is possible to use different time-tables for different seasons or months of the year. Each day a given time-table is realized, it means that at a time given by the time table a vehicle, selected from vehicles available in the central node, starts from central node and is loaded with containers addressed to each ordinary nodes included in a given route. This is done in a proportional way. When a vehicle approaches the ordinary node it is waiting in an input queue if there is any other vehicle being loaded/unload at the same time. There is only one handling point in each ordinary node. The time of loading/unloading vehicle is described by a random distribution. The containers addressed to given node are unloaded and empty space in the vehicle is filled by containers addressed to a central node. Next, the vehicle waits till the time of leaving the node (set in the time-table) is left and starts its journey to the next node. The operation is repeated in each node on the route and finally the vehicle is approaching the central node when it is fully unloaded and after it is available for the next route. The process of vehicle operation could be stopped at any moment due to a failure (described by a random process). After the failure, the vehicle waits for a maintenance crew (if there are no available due to repairing other vehicles), is being repaired (random time) and after it continues its journey. The vehicle hauling a commodity is always fully loaded or taking the last part of the commodity if it is less than its capacity.

3 DTS Formal Model

3.1 Overview

The described in the previous section regional part of the Polish Post transportation system with one central node and several ordinary nodes was a base for a definition of a formal model of a discrete transport system (*DTS*) [17]. Generally speaking users of the transport system are generating tasks which are being realized by the system. The task to be realized requires some services presented in the system. A realization of the system service needs a defined set of technical and human resources. Moreover, the operating of vehicles transporting mails between system nodes is done according to some rules – some management system. Therefore, we can model discrete transport system as a 4-tuple:

$$DTS = \langle CM, BS, TI, MS \rangle \tag{1}$$

where: *CM* – client model, *BS* – business service,
TI – technical and human infrastructure, *MS* – management system.

3.2 Technical and Human Infrastructure

During modelling of technical infrastructure we have to take into consideration functional and reliability aspects of the post transport system. Therefore, the technical infrastructure of *DTS* could be described by four elements:

$$TI = \langle No, V, MM, DM \rangle \tag{2}$$

where: *No* – set of nodes, *V* – set of vehicles, *MM* – maintenance model,
DM – driver model.
Set of nodes (*No*) consists of single central node (*CN*), a given number of ordinary nodes (ON_i). The distance between each two nodes is defined by the function:

$$distance : No \times No \rightarrow R_+ \tag{3}$$

Each node has one functional parameter the mean (modelled by normal distribution) time of loading a vehicle:

$$loading : No \rightarrow R_+ \tag{4}$$

Moreover, the central node (*CN*) has additional functional parameter: number of service points (in each ordinary node there is only one service point):

$$servicepoints : CN \rightarrow N_+ \tag{5}$$

Each vehicle is described by following functional and reliability parameters:

- mean speed of a journey

$$meanspeed : V \rightarrow R_+, \tag{6}$$

- capacity – number of containers which can be loaded

$$capacity \ : V \rightarrow R_+, \tag{7}$$

- mean time to failure

$$MTTF : V \rightarrow R_+, \tag{8}$$

  a time when failure occurs is given by exponential distribution with mean equal to a value of *MTTF* function,
- mean repair time

$$MRT \ : V \rightarrow R_+. \tag{9}$$

The traffic is modelled by a random value of vehicle speed and therefore the time of vehicle (v) going from one node (n_1) to the other (n_2) is given by a formula

$$time\ (v, n_1, n_2) = \frac{distance\ (n_1, n_2)}{Normal\ (meanspeed\ (v), 0.1 \cdot meanspeed\ (v))}. \tag{10}$$

where *Normal* denotes a random value with the Gaussian distribution.

Maintains model (*MM)* consists of a set of maintenance crews which are identical and unrecognized. The crews are not combined to any node, are not combined to any route, they operate in the whole system and are described only by the number of them. The time when a vehicle is repaired is equal to the time of waiting for a free maintains crew (if all crews involved into maintenance procedures) and the time of a vehicle repair which is a random value with the Gaussian distribution ($Normal(MRT(v), 0.1 \cdot MRT(v))$. The human infrastructure is composed by the set of drivers. So the description of this part of system infrastructure requires the analysis of the drivers' state and the algorithms, which model the rules of their work. Each driver could be in one of following states (s_d):

- rest (not at work),
- unavailable (illness, vacation, etc.),
- available (at work – ready to start driving),
- break (during driving),
- driving.

The number of driver working hours is limited by the labour law. For analysed Post Transportation System the daily limit for each driver equals to 8 hours and a single driver operates with one truck. It gives a simple algorithm:

- if $w_h > limit$ then $state$ = "rest" & $w_h = 0$,
- where w_h - working hours
  $limit$ = 8 hours

Drivers in Polish Post works in two shifts, morning or afternoon one. So twice a day a driver state and shift type is analysed:

- at 6am for each driver:
  - if $shift$ == morning & s_d == rest then s_d = available,
- at 1pm for each driver:
  - if $shift$ == afternoon & s_d == rest then s_d = available,

The next problem ought to be modelled is the driver's illness state. We propose the following approach:

- for every driver at 4am:
  - if s_d == rest and $rand() < d_i$ then during x days (according to a given distribution) the driver is in s_d = unavailable,
    where d_i – driver's illness parameter.

Moreover we propose to categorise the driver's illnesses as follows:

- short sick: 1 to 3 days,
- typical illness: 7 to 10 days,
- long-term illness: 10 to 300 days.

We record the daily story of the driver. The algorithm to fix the driver to the vehicle is the last part of the driver model:

- if no driver – the vehicle does not start,
- driver can be chosen if: s_d = available
  and $w_h + estimated\ time\ of\ journey < limit * 1.1$,
- the driver is chosen randomly or by least value approach:
  abs(limit – w_h - $estimated\ time\ of\ journey$).

3.3 Business Service

Business service (*BS*) is a set of services based on business logic, that can be loaded and repeatedly used for concrete business handling process. Business service can be seen as a set of service components and tasks, that are used to provide service in accordance with business logic for this process. Each service component in *DTS* consists of a task of delivering a container from a source node to the destination one.

3.4 Client Model

The service realised by the clients of the transport system is sending mails from a source node to a destination one. Client model consist of a set of clients (*C*). Each client is allocated in one of nodes of the transport system:

$$allocation : C \rightarrow No . \tag{11}$$

A client allocated in an ordinary node is generating containers (since, we have decided to monitor containers not separate mails during simulation) according to the Poisson process with destination address set to ordinary nodes. In the central node, there is a set of clients, one for each ordinary node. Each client generates containers by a separate Poisson process and is described by intensity of container generation:

$$intensity : C \rightarrow R_{+} . \tag{12}$$

The central node is the destination address for all containers generated in ordinary nodes.

3.5 Management Model

The main goal of management model is to control the movement of trucks. As it was mentioned in section 2, the Polish Post transportation system is ruled by time-tables. However, in [19] we proposed a heuristic management approach to replace the time-table based solution. The substitution is necessary for automatic analysis of model system metrics in a function of system parameters. Changes in system parameters like a number of trucks or a number of drivers require tuning of the time-table. This is hard to be realized in automatic way and is very time consuming. Moreover, we have shown in [21] that heuristic algorithm similar to those presented here is more effective in a usage of the vehicles (requires less vehicles then time-table approach to achieve availability of the system on a given level) and to allows to react for the critical situations which can occur during the normal system work. The decisions (send a truck to a given destination node) are taken in moments when a container arrives to the central node. The truck is send to a trip if:

- the number of containers waiting in for delivery in the central node of the same destination address as that just arrived is larger than a given number,
- there is at least one available vehicle,
- the simulated time is between 6 am and 22 pm minus the average time of going to and returning from the destination node.

The truck is send to a node defined by destination address of just arrived container. If there is more than one vehicle available in the central node, the vehicle with size that a fits the best to the number of available containers is selected, i.e. the largest vehicle that could be fully loaded. If there are several trucks with the same capacity available the selection is done randomly. On the other hand we observe in the same way the vehicles available in the ordinary nodes. The only difference is the greater level of threshold to initialise the vehicle journey. The

restriction for the time of truck scheduling (the last point in the above algorithm) are set to model the fact that drivers are working on two 8 hours shifts.

4 Availability Metric

Dependability analysis is based on the assessment of some performance measures. We propose to use on of dependability metric: availability (readiness for correct service), for *DTS* analysis. In general, availability is defined as the probability that the system is operational (provides correct responses to the client requests) at a specific time. In stationary conditions, most interesting from the practical point of view, the function is time invariant. Thus, the availability is characterized by a constant coefficient.

In the previous sections we have stated that the main goal of the system is to deliver the mail on-time. Therefore, we define the operational state of the *DTS*, as a state when there are less than a defined limit delayed containers in the system. To define the delayed container, let us, introduce the following notation:

- T – a time measured from the moment when the container was introduced to the system to the moment when the container was transferred to the destination (random value),
- T_g – a guaranteed time of delivery, if exceeded the container is delayed.

A very often used estimation of the availability, which uses its asymptotic property and is based on an assumption of a uniform rate of client requests is the acceptance ratio. For *DTS*, we have defined it [16] as the ratio of on-time containers (containers for which $T < T_g$) to all containers within a given time of observation $(0, \tau)$. Within the time period a given number of containers are delivered ($N_{delivered}(\tau)$), a part of them or all delivered on time ($N_{ontime}(\tau)$), but at the end of analysed period time there could be some containers not yet delivered (waiting in the source node or being transported) ($N_{insystem}(\tau)$) and all or part of them being not late yet ($N_{ontimeinsystem}(\tau)$). Taking into consideration introduced symbols the availability could be calculated as the expected value (Monte-Carlo approach) of ratio of on-time containers to all containers:

$$AR_\tau = E\left(\frac{N_{ontime}(\tau) + N_{ontimeinsystem}(\tau)}{N_{delivered}(\tau) + N_{insystem}(\tau)}\right). \tag{13}$$

5 Case Study Analysis

5.1 Exemplar DTS

We propose for the case study analysis an exemplar *DTS* based on Polish Post regional centre in Wroclaw. We have modelled a system consisting of one central

node (Wroclaw regional centre) and twenty two other nodes - cities where there are local post distribution points in Dolny Slask Province [18]. We have fixed the most important reliability and functional parameters of the key elements of the system. The length of roads were set according to real road distances between cities used in the analysed case study. The intensity of generation of containers for all destinations were set to 4.16 per hour in each direction giving in average 4400 containers to be transported each day. The vehicles speed was modelled by Gaussian distribution with 50 km/h of mean value and 5 km/h of standard deviation. The average loading time was equal to 5 minutes. There was single type of vehicle with capacity of 10 containers. The MTF of each vehicle was set to 20000. The average repair time was set to 5h (Gaussian distribution). We also have tried to model the drivers availability parameters. We have fulfilled this challenge by using the following probability of a given type of sickness:

- short sick: 0.003,
- typical illness: 0.001,
- long-term illness: 0.00025.

We hope. that the proposed approach can properly model the real problems with a driver availability at transportation enterprises.

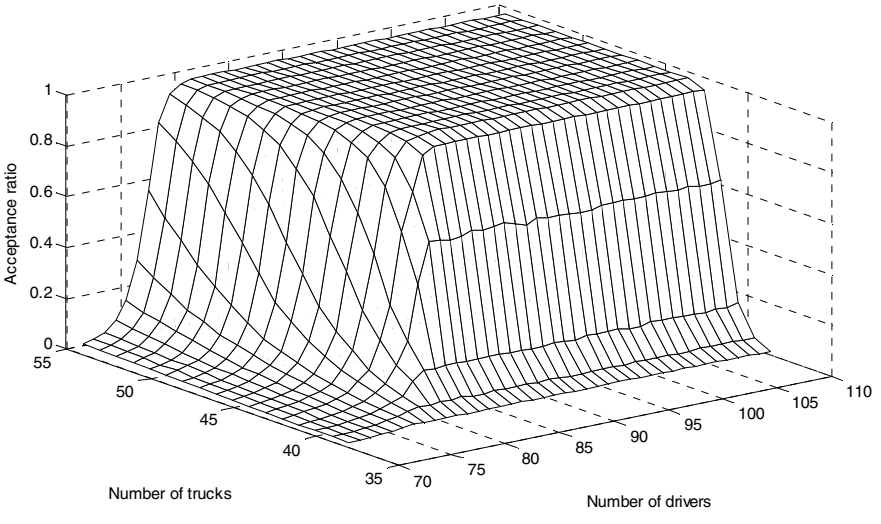


Fig. 1 Acceptance ratio in a function of number of trucks and number of drivers for tested *DTS*

5.2 Results and Discussion

We have simulated the system for tasks defined above using 70 to 100 drivers and 38 to 54 vehicles. The results, the acceptance ratio, are presented in Fig. 1. We can

easily observe the kind of the pareto set: the bordered numbers of drivers and vehicles which guarantee the acceptance ration of the system at acceptable level. It is a quite natural that the owner of *DTS* is looking for the metric equal to almost one. This way he or she can say that the tasks potentially served by system can be done with no problems. On the other hand the owner can predict the critical situations: he or she can perfectly know if – for example – catastrophic failures of the vehicles start to generate the real problem for the *DTS*. The same kind of analysis can be done for a pandemic illness touching the drivers. The results generated for our exemplar *DTS* show that we need not less than 85 drivers and not less than 45 vehicles to operate in the effective way. The second test-bed was focused on the importance of the human-factor. We fixed the number of drivers and we tried to improve the acceptance ratio only by the number of vehicles. The results (Fig. 2) show that the small number of drivers cannot be substituted by the larger number of vehicles. It is easy to notice that 78 drivers is to small number to guarantee the acceptance ration at the level of 1. One more driver makes the situation better, but the number of necessary vehicles is rather large. The best solution is given for 82 drivers. The results of this kind of analysis allow to make the strategic decisions – if the next trucks ought to be introduced to the transportation fleet for example. The last set of the experiments was related to the probability density function of acceptance ratio. Since acceptance ratio is an average of a ratio of on-time containers to all containers (as defined in [13]) one could want to see this ratio distribution. It is presented in Fig. 3 for 46 trucks and four different values of number of drivers: 75, 78, 79 and 81 respectively.

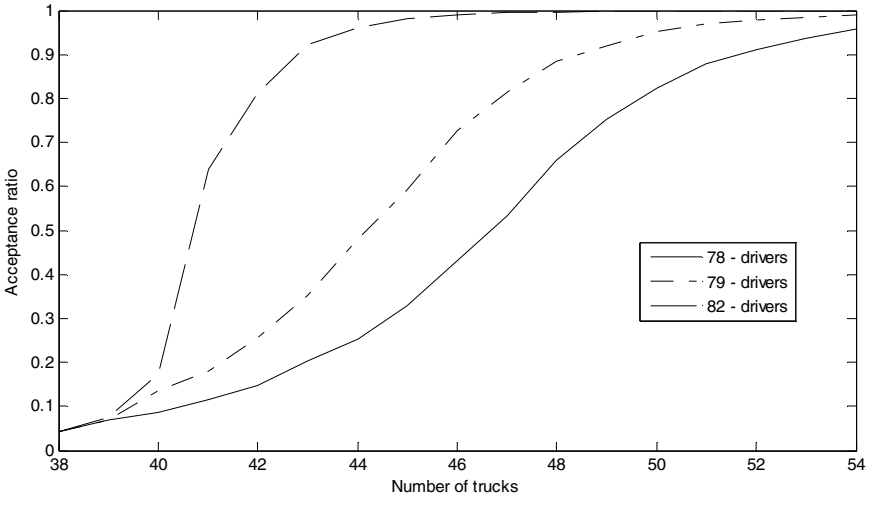


Fig. 2 Acceptance ratio in a function of number of trucks for fixed number of drivers

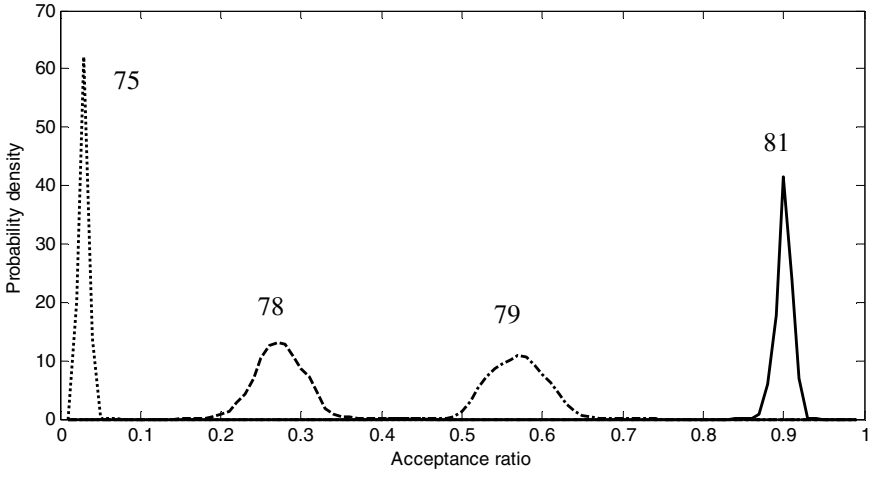


Fig. 3 Probability density function of acceptance ratio for 46 trucks and 75, 78,79 and 81 drivers

6 Conclusion

We have presented a formal model of discrete transport system (*DTS*). The *DTS* model is based on Polish Post regional transport system. The proposed approach allows to perform dependability analysis of the *DTS*, for example:

- determine what will cause a ”local” change in the system,
- make experiments in case of increasing number of containers per day incoming to system,
- identify weak point of the system by comparing few its configuration,
- better understand how the system behaves,
- foresee changes caused by human resource influence.

Based on the results of simulation it is possible to create different metrics to analyse the system in case of reliability, functional and economic case. The acceptance ratio of the system was introduced - defined in a functional way as an average of a ratio of on-time containers to all containers.

The metric could be analysed as a function of different essential functional and reliability parameters of *DTS*. Also the system could be analyse in case of some critical situation (like for example a few day tie-up [16]). The chapter includes some exemplar systems, based on real Polish Post Wroclaw area, and calculated metric.

The developed *DTS* simulator [17] makes it a practical tool for defining an organization of vehicle maintenance and transport system logistics.

Acknowledgment. Work reported in this paper was sponsored by a grant No. N N509 496238, (years: 2010-2013) from the Polish Ministry of Science and Higher Education.

References

[1] Barcelo, J., Codina, E., Casas, J., Ferrer, J.L., Garcia, D.: Microscopic Traffic Simulation: a Tool for the Design, Analysis And Evaluation Of Intelligent Transport Systems. Journal of Intelligent and Robotic Systems: Theory and Applications 41, 173–203 (2005)

[2] Barlow, R., Proschan, F.: Mathematical Theory of Reliability. Society for Industrial and Applied Mathematics, Philadelphia (1996)

[3] Ben-Akiva, M., Cuneo, D., Hasan, M., Jha, M., Yang, Q.: Evaluation of Freeway Control Using a Microscopic Simulation Laboratory. Transportation Research, Part C (Emerging Technologies) 11C, 29–50 (2003)

[4] Birta, L., Arbez, G.: Modelling and Simulation: Exploring Dynamic System Behaviour. Springer, Heidelberg (2007)

[5] Burt, C.N., Caccetta, L.: Match Factor for Heterogeneous Truck and Loader Fleets. International Journal of Mining, Reclamation and Environment 21, 262–270 (2007)

[6] Duinkerken, M.B., Dekker, R., Kurstjens, S.T.G.L., Ottjes, J.A., Dellaert, N.P.: Comparing Transportation Systems for Inter-Terminal Transport at the Maasvlakte Container Terminals. OR Spectrum 28, 469–493 (2006)

[7] Fishman, G.: Monte Carlo: Concepts, Algorithms, and Applications. Springer, Heidelberg (1996)

[8] Gartner, N., Messer, C.J., Rathi, A.K.: Traffic Flow Theory and Characteristics. In: Board, T.R. (ed.) University of Texas at Austin, Texas (1998)

[9] Gold, N., Knight, C., Mohan, A., Munro, M.: Understanding service-oriented software. IEEE Software 21, 71–77 (2004)

[10] Ioannou, P.A.: Intelligent Freight Transportation. Taylor and Francis Group, Carolina (2008)

[11] Krzyzanowska, J.: The Impact of Mixed Fleet Hauling on Mining Operations at Venetia Mine. Journal of The South African Institute of Mining and Metallurgy 107, 215–224 (2007)

[12] Liu, H., Chu, L., Recker, W.: Performance Evaluation of ITS Strategies Using Microscopic Simulation. In: Proceedings of the 7th International IEEE Conference on Intelligent Transportation Systems, pp. 255–270 (2004)

[13] Sanso, B., Milot, L.: Performability of a Congested Urban-Transportation Network when Accident Information is Available. Transportation Science 33(1), 10–21 (1999)

[14] Taylor, M.A.P., Woolley, J.E., Zito, R.: Integration of the Global Positioning System and Geographical Information Systems for Traffic Congestion Studies. Transportation Research, Part C (Emerging Technologies) 8C, 257–285 (2000)

[15] Vis, I.F.A.: Survey of Research in the Design and Control of Automated Guided Vehicle Systems. European Journal of Operational Research 170, 677–709 (2006)

[16] Walkowiak, T., Mazurkiewicz, J.: Analysis of Critical Situations in Discrete Transport Systems. In: Proceedings of International Conference on Dependability of Computer Systems, Brunow, Poland, June 30-July 2, pp. 364–371. IEEE Computer Society Press, Los Alamitos (2009)

[17] Walkowiak, T., Mazurkiewicz, J.: Availability of Discrete Transport System Simulated by SSF Tool. In: Proceedings of International Conference on Dependability of Computer Systems, Szklarska Poreba, Poland, pp. 430–437. IEEE Computer Society Press, Los Alamitos (2008)

[18] Walkowiak, T., Mazurkiewicz, J.: Functional Availability Analysis of Discrete Transport System Realized by SSF Simulator. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 671–678. Springer, Heidelberg (2008)

[19] Walkowiak, T., Mazurkiewicz, J.: Algorithmic Approach to Vehicle Dispatching in Discrete Transport Systems. In: Sugier, J., et al. (eds.) Technical approach to dependability, Wroclaw: Oficyna Wydawnicza Politechniki Wroclawskiej, pp. 173–188 (2010)

[20] Walkowiak, T., Mazurkiewicz, J.: Functional Availability Analysis of Discrete Transport System Simulated by SSF Tool. International Journal of Critical Computer-Based Systems 1(1-3), 255–266 (2010)

[21] Walkowiak, T., Mazurkiewicz, J.: Soft Computing Approach to Discrete Transport System Management. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010. LNCS, vol. 6114, pp. 675–682. Springer, Heidelberg (2010)

An Effective Learning Environment

Marek Woda[1] and Konrad Kubacki-Gorwecki[2]

[1] Institute of Computer Engineering, Control and Robotics, Wroclaw University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mail: marek.woda@pwr.wroc.pl
[2] Institute of Computer Engineering, Control and Robotics, Wroclaw University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mail: k.kubacki.gorwecki@gmail.com

Abstract. This chapter is dedicated to an effective learning environment implementation, and more precisely the development of algorithms that improve effectiveness of the learning process using the automatic selection of the difficulty of lessons depending on individual student characteristics and selection of the architecture to increase the technical efficiency of the process through the use of appropriate technical means to build e-learning system.

1 Introduction

Quite new domain of science called e-learning, has already shown so far its great usefulness. However, it is still crippled by: the lack of a unified, coherent vision of e-education. In most cases, the theory has not been matched with a practice, and so it is in the case of distance education. In e-learning emphasis is placed on theory of teaching, but from a technological side there neither limitations nor strict guidelines, which in many cases adversely affect the operation of newly developed systems. Currently, the most of teachers recognizes the need to standardize and streamline the process of that kind of education. Increasingly, one talks about standardization of e-learning systems.

Teaching effectiveness is a function of variety of forms, methods and means of teaching [12]. Better efficiency of knowledge acquisition in the era of computerization and surrounding flood of information can only be achieved by the proper use of technical means combined with classical forms of teaching methods, leading to the construction of educational structures integrated with modern technologies and means of formal presentation [14].In the text Authors should use terms *chapter*, *section* or *work* instead of *paper*.

Teaching methodology [13] is an interdisciplinary study on the effectiveness of education, seeking to answer the question of how to teach and learn faster, better and cheaper in certain circumstances. The interdisciplinary nature of this technology lies in the fact that it draws the object of interest and research methods from other disciplines as computer science, cybernetics, systems theory and the theory of communication.

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 285 – 299.
springerlink.com

A large computational complexity of e-education algorithms, as well as a large number of data subject to analysis, influences the course supervisors with plethora of task [26], due to insufficient resources required for routine maintenance tasks. The number of people involved in a control, planning, scheduling classes, and time to assess the progress of listeners grows in proportion to the number of listeners. In the traditional remote learning all of the processes are carried out by the "human factor" [5, 11, 13], which can cause (and usually do) issues like a loss of control over during the learning progress, decreased interest among course participants (caused by “lecturer fatigue”), and teaching contact incongruity to the maturity / skills of listeners, which equals to the loss of their interest / willingness to acquire knowledge, etc. However, excluding completely human factor from the teaching process is unfavorable [10, 19, 20] too, it leads among other things to a sense of alienation and lack of students control.

A remedy to the disadvantages outlined above, and a way to increase the efficiency and effectiveness of the process of acquiring knowledge may be an intelligent agent based system [1, 3, 4].

In the era of dynamic Internet growth and widely accessible broadband access, more and more institutions see a potential use of these resources to provide educational services [61].

Despite the existence of a multitude of both commercial (JoomlaLMS, WEBLessons, Blackboard Learn) and open platforms (LAMS, RCampus, Moodle) of distance learning, most of them work as merely as a knowledge container, and provide only simple mechanisms for data sharing [6, 7, 23]. A foundation of these systems constitutes combination of different electronic media [49]. Mainly because of it, the basic idea of teaching was lost; namely, focus on a student and its individual characteristics. The fundamental problem in current systems is the lack of a model student, and individualization of the learning process so that was the most suited to individual personal characteristics [10].

Teaching theory suggests [15] that the efficiency of the process of acquiring knowledge is based on an adaptation absorption orchestrated by individual student characteristics. Based on our own experience we can distinguish, at least a few factors that determine the effectiveness of learning. These include:

- Complexity of the material
- Quantity and diversity of content
- Duration of lessons
- Time to acquire knowledge

It was noticed [23] that it would be a huge leverage if these aspects could be taken into account as factors adjusted to the characteristics of individual participant, and it would result in significantly increased efficiency of the process.

Practical application of distance learning on the Internet requires implementation of special software called e-learning platforms. There are numerous, such kind, platforms that support remote learning process [6,7].

All platforms have more or less complex functions associated with managing the learning process [22, 23, 26]. Current commercial solutions that can be found on a market - suffer from the childhood imperfections – usually e-learning platform

evolved from content management systems, which was not developed with the idea of personalized knowledge delivery. Many new features, from a pedagogical point of view have been missed - the most troublesome is the lack of student diversification. Each one is treated the same, unified way, which prevents screening for outstanding students as well as those who cannot cope alone with the material.

In the most of distance learning systems focus on the technical aspects of teaching, and omitted or treated perfunctorily an extremely important aspect of education- entity oriented teaching. Follow worthy idea is to create student models [11], and based on them implement adaptive systems.

Many papers [17, 18, 19] highlights the lack of comprehensive approach to e-education, among other things to focus on delivering educational content with exclusion of personal characteristics or individual attention to students. One should seek opportunities to combine modern techniques with the appropriate modeling of students with different skills and levels of expertise.

A system, which would combine the adaptation of learning material with level of expertise of individual student along with controlling the progress of science, and delivering relevant content, is still missing. Emerging systems should be based primarily on well-built model of the student and be self-adapted systems to encompass the needs of individuals.

This work seeks to connect these demands in a single system, which will form the basis for the modular e-learning system, designed to respond to the shortcomings of current solutions.

2 System Architecture

In this chapter a novel concept of agent based e-learning system that supports e-learners supervision in a distance education is presented. The presented system model implements teaching strategies described in details in [22, 24].

2.1 Basic Terms and Assumptions

This section presents the basic terms and assumptions used in the description of the agent based system concept.

The supervision of distance learning, in the context of the considered system concept, is understood as:

- monitoring, track learning progress of the each course participant and reporting it to the course coordinator
- optimizing student's learning path, based on information about learning style preferences (VAK model [25]); its progress and performance in knowledge acquisition, in such a way as to maximize average of the grades for all course participants
- preparation an individual learning pace and repetition schedule for all lesson units of a course for all course participants
- ability to search for additional related educational materials and share them with other participants

E-learning system is an agent based system that consists of the following basic elements:

- a knowledge base, accumulated in a course, represented by a course knowledge graph
- competence and self-assessment tests, quizzes
- historical data that show progress of a student education
- initial data collected after course qualification test that discloses student type (e.g. VAK test [25])
- a set of learning paths.

Domain knowledge in the system is represented by a directed graph, with no recurrence. The knowledge is divided and contained in lesson units (composed of knowledge quanta). Learning process of each student is monitored and the knowledge acquired in this process is subject to evaluation by both self-assessment and final tests. Lesson units in the system are represented by three different levels of difficulty (basic, intermediate, advanced) and each one with a different style of course content presentation; the content difficulty levels directly correspond to student model categories. It is assumed that to complete a part of material represented by a course, it must exist at least one lesson, which provides this content for any given level of difficulty, and at least three forms of content presentation. In principle, each quantum of knowledge has at least 9 lesson units containing it (3 levels of difficulty, three styles of content presentation.)

Course knowledge graph (Fig. 1) is a directed graph is a hierarchical structure. The edges of the graph indicate possible directions of movement between the vertices. Vertices are treated as individual lessons or evaluation points. It is assumed that the graph has a starting point - the preliminary competency test, and the final element - course ending test of competence.

A **learning path** is the sub-graph of course knowledge graph that contains defined minimum number of vertices (to pass by a student) required to finalize a course. The transition between two nodes in the path is always in a fixed direction, and allowed only after positive assessment of knowledge acquired in the previous node. It is assumed that completion of a path containing only lesson units at basic difficulty level is sufficient to obtain a minimum level of knowledge to pass a course.

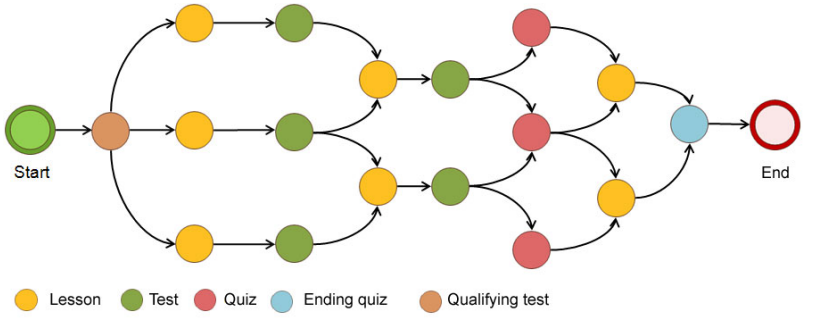


Fig. 1 Example of course knowledge graph

A **course content presentation strategy** is a method of adapting the presentation of course content that should strive to match the individual student's learning style.

A **teaching strategy** is a method to adapt the course to a level of intellectual capacity and interests of a student. Any strategy assumes that the student should realize the highest possible level of lesson difficulty allowing him to gain positive results during the test. A strategy strives ensuring that the student's level of knowledge after completing the course will reach the maximum. The strategy is subject to the evaluation of during learning progress in the real time, and if necessary - is adjusted for better results.

Student profile represents a vector characterized by psychological and intellectual profile of a student. It includes information about the preferred student learning style, the general level of student knowledge, recent results; preliminary results from the course competence test, and detailed test results that make possible to categorize student's type.

2.2 The Concept of Agent Based System

Developed by the authors the concept of remote monitoring system to support learning activity involves the following agents:

- User Agent (UA)
- Learning Path Agent (LPA) - builds a learning path for students
- Schedule Builder Agent (SBA) - creates a learning schedule
- Learning Style Agent (LSA)
- Computing Agent (CA)
- Search Agent (SA) searches for a content related data during learning progress

Relations between agents in the system are depicted in the diagram (Fig. 2).

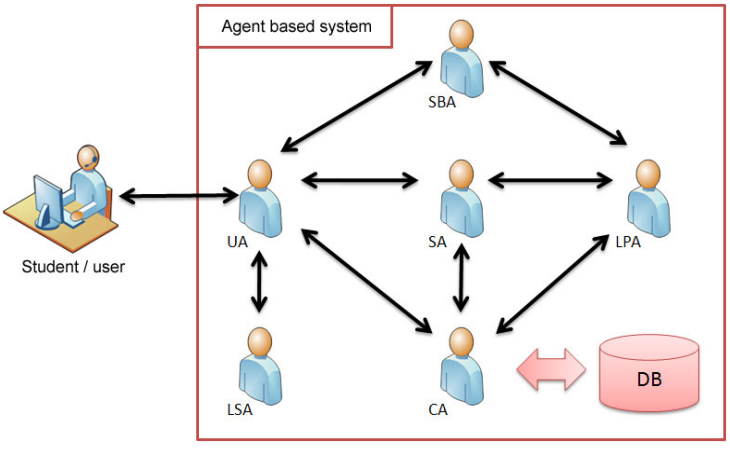


Fig. 2 Relations and interactions between agents in the system

2.2.1 User Agent

User Agent (UA) is responsible for a direct communication and cooperation with a user (student / course participant). Its main tasks include the collection of data on user activity - logon time, how much time one spends online, which course elements are explored etc. This agent is also responsible for presenting a learning schedule to the user, imposing learning pace, and processing and storing recent tests results as well. It acts as a proxy in a communication with other system agents – acting as a specific type of intermediary (an interpreter) between the agent based system and a user.

Each student in the system, at the time of signing in, is assigned an individual UA agent. This agent operates as long as its user is logged in the system. From other agents in the system point of view, an AU represents (and reflects) a user behavior.

2.2.2 Learning Path Agent

Learning Path Agent (LPA) builds a learning path for students, and is the most important agent in the system. Its mission is to prepare an optimal learning path for each student. During learning path creation LPA gets defined in the database course structure and based on it creates a knowledge course graph. Next having analyzed the student learning preferences, and his learning performance, it chooses appropriate learning path, satisfying two conditions:

1. Select learning path that match the preferred learning style best
2. Select the most advanced track (difficult) that student can pass (based on historical data if available), and at the same time giving a student an opportunity for performance incentives to work.

In the system, depending on the number of students enrolled in courses they may operate one or more LPA agents. Please note that the tasks carried out by LPAs may need a lot of computation time) and may use too much time, this is proportionate to the complexity of the knowledge graph. Algorithms used for learning path building have been described in detail in Section 3.

2.2.3 Schedule Builder Agent

Schedule Builder Agent (SBA) creates a learning schedule; it operates based on results attained by a LPA agent. Its functionality is to create a timetable with a schedule how the lesson units should be learnt and the knowledge acquired evaluated – this is being set in the rate imposed by the course developer.

Initially, the schedule is arranged based on the student's educational path (Fig. 3). In the example below, we can see planed accordingly: Lesson 1, Lesson 2, a control test for Lesson 1 and 2, Quiz 1, Lesson 3, Lesson 4, the control test after Lesson 3 and 4, and final exam at the end of learning path.

Fig. 3 Initial realization schedule of a learning path

During the execution of scheduled tasks, after control test [T1, 2] may turn out that the student has not mastered the material from Lesson 2. The task of the SBA is to plan repetition of this unit before the student, takes a final quiz covering this material, whose results affect the final student's evaluation (Fig. 4).

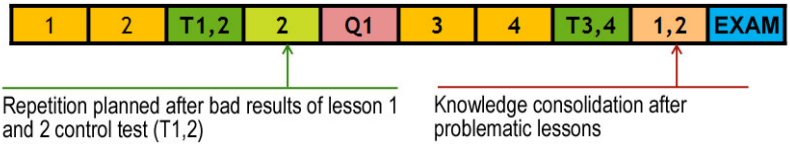


Fig. 4 Changes in the schedule after first two lessons

On the given above, Figure 4, one can notice that a re-iteration of the material from Lesson 2 is scheduled directly by the quiz [Q1] and a knowledge consolidation of Lesson 1 and 2 just before the final exam. This schedule could change if the student still has the same problems with mastery of the content of lessons of 3 or 4, and if he fails to pass quizzes or exams.

The student, himself, is also able to change his schedule - it can either resign from the planned units or propose to repeat those for which is not sure that yet he mastered.

2.2.4 Learning Style Agent

Learning style agent (LSA) is responsible for correlation a learning content with student preferred learning style, and also for updating the student profile in terms of those preferences.

The most important task of LSA is to evaluate the form and deliver adequate form of presentation of a lesson unit that matches the student's type. This evaluation takes into account three aspects:

1. Preferred student learning style
2. VAK test results
3. Meta data describing the contents of a teaching unit (SCORM object)

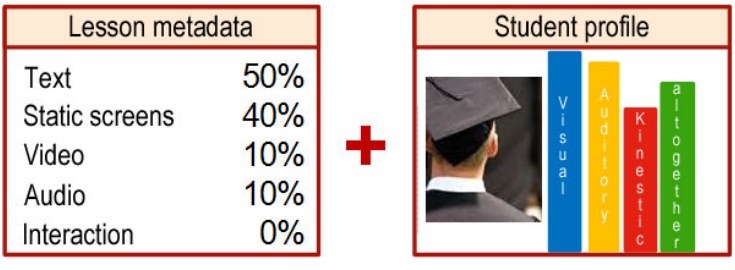


Fig. 5 Parameters taken into account during selection of lesson style representation

It is assumed that the ideal situation is when all education objects consist of information clearly indicating which the student styles (e.g. from VAK model

[8, 25]) a particular lesson representation fits best. This information is not given in advance, so we have to propose a solution based on the information that is already at hand, allowing discovery the right student style.

The solution is available with each lesson, and is stored in its metadata. Metadata contain information about what didactical material lesson units is built of. If this type of material as presented on Fig. 5, we can define a function that allows us to match a style with a lesson unit. The rules are fairly simple. Classification relies on the attribution of specific content types to a student style characterized by VAK model, and assigning those appropriate values:

1. Visual learners: text, image, video
2. Auditory: video, audio
3. Kinetics: video, audio, interaction

How to precisely address this issue, we are about to demonstrate in section 3.

2.2.5 Calculation Agent

Calculation Agent (CA) is responsible for interaction with external systems. In this case the agent communicates with the course database. Its tasks include the data registration received from other agents: UA (login, timeout), LPA (generated learning path), LSA (student profile), SBA (record of the learning schedule) and providing the necessary information at their requests.

There can be more than one CA agent in the system. Their number depends on the number of students, the number of database engines working in the system, and the maximum number of connections to servers. Too small number of CA agents becomes quickly usually a bottleneck for the entire agent based system.

So why shall one implement this kind of agent? Its presence in the system allows you to operate on multiple nodes of distributed databases. Thanks to that the database holding learning objects can be located on a different machine than the database with user data. It also makes possible interaction with other instances of the system in a completely transparent way. For the remaining agents in the system, there is no access to the database. They can only notify CA about the demand for data access.

2.2.6 Searching Agent

Searching agent (SA) is a classic element given in the agent systems literature. Its goal is acquisition and delivery of supporting material related to teaching content from various sources and systems, without user knowledge, based on defined search criteria like:

1. Keywords
2. Materials that extend the content of a lesson unit
3. Teaching materials that meet the learning style criterion

Searching agents immediately after the spawn, start browsing the yellow pages service of JADE environment for all the agents that can provide them with learning materials for the selected criteria, and go into sleep mode if are no longer in

use. When a user or an agent (e.g. LSA, LPA) starts looking for related materials, an SA agent will query known data sources for the materials that meet the criteria, and submit the results to a user / agent.

3 Adaptation Algorithms

In the previous section we discussed the characteristics of particular agents in the designed system. Most of them are not sophisticated and do not require complex logic to run them; They operate based on communication with the environment, and performing simple tasks - displaying, storing and retrieving data in the database, building queries, or scheduling tasks. Two agents working in the system - LSA and LPA - implement a way more, complex functionality, details of which will be presented in the following section.

3.1 Learning Path Building

Strategic stage, from the perspective of presented system concept, is optimal learning path building that matches each enrolled student. This process consists of two main stages - building a course knowledge graph, and building learning path for a student, which is a course knowledge sub-graph. Let us discuss in detail each of these stages.

3.1.1 Stage 1

Course knowledge graph is being re-built every time when its structure changes. In most cases, when we are dealing with the mature course (tested and used before), this process must be to done only once, right after the course has been created (started) - and then its structure is saved.

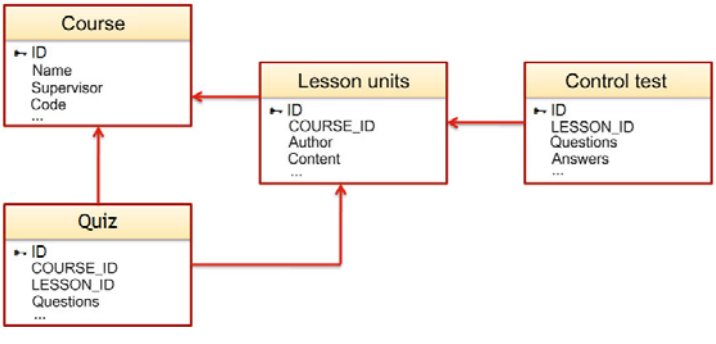


Fig. 6 Example course structure DB schema

Construction of the graph starts from collection from the database information about the structure of the course. To this end, LPA agent that builds it, sends a request “course structure” query to a CA agent, which has access to the database,

and it sends back the result (Fig. 6). Based on the feedback received, the LPA builds a directed graph structure. This process is briefly described at Listing 1.

Listing 1 Course knowledge graph building

```
public class CourseGraph implements Serializable {
        ...
      public CourseGraph () {
             graph = new DirectedGraph<LessonUnit,
                              LearningArc<LessonUnit>>();
              }

      public void bulidGraph(List<LessonUnit> objectsList,
                                    Student student) {
               // add vertices
               for(LessonUnit vertex : objectsList) {
                      graph.addNode(vertex);}

               // add branches
               for(LessonUnit w : objectsList) {
                   for(LessonUnit nast : w.nextt()) {
                       graph.addArc(nast, vertex, new
               WeightLessonSelection(student));
                                  }
                      }
            }
}
```

3.1.2 Stage 2

Before discussing the process of determining the optimal learning path, one have to answer the question, what is the optimal path of CKW graph? Being aware that the algorithm that will shape the path will operate on the graph structure, we can try to transpose the problem of optimality in the well-known problems known from graph theory. It is easy to see the analogy of the problem, to finding the shortest path between two vertices in the directed graph.

Therefore lets' define the optimality of the path in such a way that the weights of each arc graph - symbolize the transition between the different lesson units of the course - to express the material adaptation in the next lesson to the student's current needs (level of expertise, learning style). Transition between each lesson units is described in Listing 2.

Listing 2 Calculation of arc weights

```
const lesson_advcacement_weight= 3;

advancement_coeff := lesson_level / (|lesson_level – student.level|
+1);

weight := advancement_coeff * lesson_advcacement_weight +
          lessonAdjustmentForStudent(student, lesson);
```

The code quoted above besides the advancement level includes also student's preferred learning style. The lesson adjustment (to a student preferred learning style) process has been described in the next section 3.2.

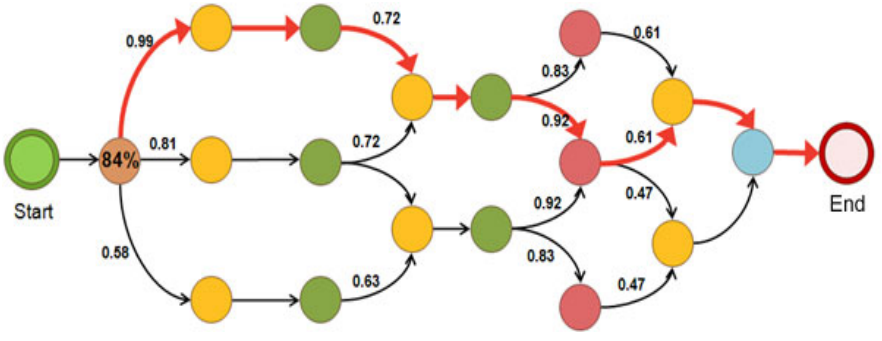


Fig. 7 Example of course knowledge graph with weights and learning path visualization

As a result of given above operation we get directed graph with no cycles with positive weights. The perfect algorithm that finds shortest path is Dijkstra's algorithm. Its computational complexity depends on the number of vertices *V* and edges *E* of the graph. Its complexity varies on priority queues implementation:

1. queue as an array - it gives the complexity of $O(V^2)$,
2. queue in the form of the mound - $O(ElogV)$

The first variant is optimal for dense graphs, the other for the rare graphs. In a typical knowledge graph it will better solution to apply the first variant, because the number of edges will be far greater than the number of vertices.

Listing 3 Detremination of optimal learning path

```
        public GraphPath computeLearningPath(Lesson start, Lesson
ending) {
        GraphPath<Lesson, LearningArc<Lesson>> path = null;
        Dijkstra dij = new
     Dijkstra<Lesson,LearningArc<Lesson>>(graf);
        path = dij.execute(start, ending);
     return path;}
```

3.2 Content Presentation Strategies

In the previous two sections, we referred to an algorithm that will determine the degree of adaptation of the lesson content to a student's preferred learning style. There are at least two ways to achieve this goal.

The first way, a trivial one, assumes that each object within the e-system will have metadata that clearly reflects one of the VAK styles. This approach, though simple (Listing 4), is not devoid of drawbacks. One of them is the necessity of human involvement to the design lesson - as though it seems intuitive, does not always have to

be. For example, the question arises to which style one shall assign didactic material presented in the form of a movie - a combination of animation, experimental video and a standard lecture. In some extent it matches all VAK styles - and depending on the assessment may be classified differently. A second drawback of this approach is revealed when there is no content for a certain learning styles, then one has to cope with a problem which material to present to the student.

Listing 4 Trivial lesson style determination

```
        public boolean doesLessonMatchLessonStyle (Student s, Lesson
l) {
        StyleLearning studentStyle = s.styleLearning();
        StyleLearning lessonStyle  = l.styleLearning();

        return studentStyle.equals(lessonStyle);}
```

The second way to achieve the objective is to identify the style of material representation in a lesson unit based of metadata, describing its contents. Taking into account the student profile, we can propose a measure called student style adjustment, which reflects the percentage of how a student's profile is consistent with the profile of lessons (Fig. 8).

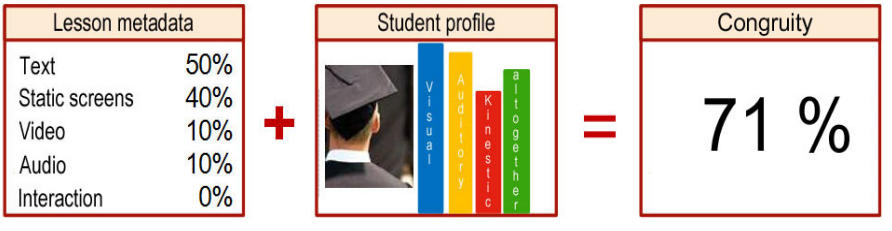


Fig. 8 Input data and result of *student style adjustment* (congruity level)

This approach allows for a better lesson unit selection, in the absence of preferred unit (one that matches exactly student's learning style). Tests have proven that method brings good results if weights for individual components are introduced (see Listing 5).

Listing 5.5 Congruity level evaluation

```
public float studentStyleAdjustment (Student, Lesson) {
   coef_vis = student.levelVisual* enforcePreference(visual) *
   (lesson.text + lesson.picture + lesson.video);
   coef_aud = student.levelAuditory * enforcePreference(audio) *
   (lesson.audio + 0.5 * lesson.video);
   coef_kin = student.levelKinestic * enforcePreference(kinest) *
   (lekcja.interAction + 0.5 * lekcja.video);
   return coef_vis + coef_aud + coef_kin; }

public float enforcePreferredLevel(Style) {
    if(style.equals(student.styleLearning)) return 10f;
    return 1f; }
```

3.2.1 Change of Presentation Strategy

It was stressed (21, 24, 25) the fact that the recognition of learning style is a difficult issue, and the same result is often ambiguous. To ensure the adaptability of the system, the results of VAK test, are evolving as one works with a user (student). The individual components that indicate compliance with one of the VAK profiles are constantly being modified, as proposed in (Formula 1).

$$W_z = 0.8 \cdot W_z + 0.2 \cdot T_i \tag{1}$$

W_z – actual discriminant value of student proficiency
T_i – i-th result of competence test

Student proficiency is determined empirically, a level of intellectual abilities of a student. On its basis, e-system decides whether the student will be routed on basic, average, or advanced learning path of the course. This discriminant is modified during the evaluation tests. Its value depends on the weighted average of its current value and the test result.

Having the information about lessons structure and test results, we can modify these three values proportionally to the test results and the amount of material characteristic for a particular learning style, contained in the lesson evaluated.

A change of presentation strategy may also occur in a more dynamic way, if the student's results are unsatisfactory. Probability of change strategy in time is given using equations (Formula 2 and 3):

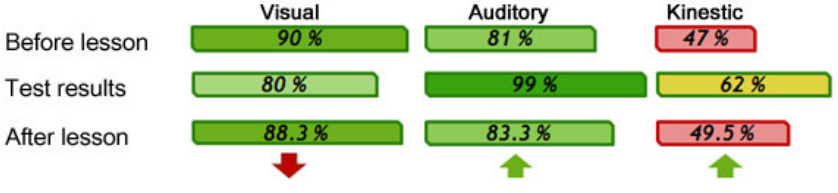


Fig. 9 Example of course knowledge graph with weights and learning path visualization

$$L_{\text{strategy change}} = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} \cdot e^{-\left(\frac{t}{\lambda}\right)^{k}} \cdot f\left(result\right) \tag{2}$$

$$f\left(result\right) = \begin{cases} 0, \text{ result} > 75\% \text{ or trend} > 0 \\ 1, \qquad\qquad \text{othercase} \end{cases} \tag{3}$$

t – time, $t > 0$
λ – scale parameter; (this case: $\lambda = 1$)
k – shape parameter; (this case: $k = 5$)

4 Conclusions

The concept of agent based e-learning system described in this chapter, is based on the idea of student's learning style detection and learning material delivery corresponding to his preferences and intellectual ability. As a student model, well-known VAK [8, 25] model was used. It divides the human population into three groups: visuals, auditoria and kinestics. Correct learning style detection may increase considerably the effectiveness of student's knowledge acquisition. Tests of presented e-system, in a simulated learning environment, have proved its usefulness. Success ratio of passing final exam increased from 89.9% to 98.7%, and generally understood the level of student knowledge raised from 7.48 to 10.38 points - the maximum score was 16 points.

The use of software agent to implement this approach makes system modular allowing for more distributed collaboration. Agents in the environment can work in a number of containers that can be run on different computers within a network (in particular Internet). The benefits are: the ability to freely exchange data between the independent e-learning platforms, which include both teaching content - which is also often discussed in the literature [5, 16] – and all students as well, including historical data related to learning style, ongoing courses, general intellectual profile etc. This is not yet approach that brings many benefits and risks too.

References

[1] Baloian, N., Motelet, O., Pino, J.: Collaborative Authoring, Use and Reuse of Learning Materi-al in a Computer-integrated Classroom. In: Procs. of the CRIGW 2003, France (2003)

[2] Bacopo, A.: Shaping Learning Adaptive Technologies for Teachers: a Proposal for an Adaptive Learning Management System. In: 4th IEEE International Conference on Advanced Learning Technologies (2004)

[3] Capusano, N., Marsella, M., Salerno, S.: An agent based Intelligent Tutoring System for distance learning. In: Proc. of the International Workshop on Adaptive and Intelligent Web-Based Education Systems, ITS 2000 (2000)

[4] Dinosereanu, M., Salomie, I.: Mobile Agent Solutions for Student Assessment in Virtual Learning Environments. In: IAWTIC 2003, Austria (2003)

[5] Bitonto, D.I.: Multi-agent Architecture for Retrievieng and Tailor Los in SCORM Compilant Dis-tance Learning Environment. In: Rosson, M.B. (ed.) Advances in Learning Processes Book. InTech (January 2010), ISBN 978-953-7619-56-5

[6] EduTools. CMS: Product Comparison System (June 11, 2006) http://www.edutools.info/compare.jsp?pj=8&i=263,276,299,358,366,386,387

[7] EduTools. CMS: Product List (June 2, 2006), http://www.edutools.info/item_list.jsp?pj=8

[8] Vark, F.N.: Questionnaire, http://www.vark-learn.com/english/page.asp?p=questionnaire (accessed December 2010)

[9] Grob, H.L., Bensberg, F., Dewanto, B.L.: Developing, deploying, using and evaluating an open source learning management systems. In: 26th International Conference, Information Technology Interfaces (2004)
[10] Jones, D., Behrens, S.: Online assignment management - an evolutionary tale. In: 36th Annual Inter-national Conference on System Sciences. IEEE, Los Alamitos (2003)
[11] Kavcic, A.: The role of user models in adaptive hypermedia systems. In: Electrotechnical Conference MELECON 2000 (2000)
[12] Leja, K. M.: Efektywność i jakość w działalności szkoły wyższej na przykładzie wybranych uczelni technicznych. PhD thesis, Department of Management and Economy, Gdansk University of Technology (2000) (in Polish)
[13] Mortaglia, T.: An automatic evaluation system for technical education at the University level. IEEE Transactions on Education (2002)
[14] Narsingh, D.: Graph theory with applications to engineering and computer science. Pretince Hall, New Jersey
[15] Koper, R., Tattersall, C.: Learning Design: A handbook on modeling and delivering networked education and training. Springer, Heidelberg (2005)
[16] Orzechowski, T.: The Use Of Multi-agents' System. In: Soomro, S. (ed.) E-learning Platforms E-learning Experiences and Future, Book. InTech (April 2010), ISBN: 978-953-307-092-6
[17] Osiński Z.: Możliwości, jakie stwarzają platformy e-Learning w edukacji, Materiały z V Ogólno-polskiego Forum Koordynatorów Technologii Informacyjnej, Mielec, Poland (2003) (in Polish)
[18] Rosenberg, M.J.: E-Learning: strategies for delivering knowledge in digital age. McGraw-Hill, New York (2001)
[19] SCHUTTE J.G.: Virtual Teaching in Higher Education: The New Intellectual Superhighway or Just another Traffic Jam? (2001), http://www.csun.edu/sociology/virexp.htm
[20] Tadeusiewicz, R., Chrząszcz A., Gaś, P., Kusiak, J.: Współpraca między uczniem a nauczycielem w nauczaniu wspomaganym przez Internet.In: Mischke, J. (red.)Akademia on-line, WSHE Łódź (2005) (in Polish)
[21] Woda, M.: Concept of composing learning content into learning tree to ensure reliability of learning material. In: Proceedings of International Conference on Dependability of Computer Systems, DepCoS - RELCOMEX 2006, pp. 374–381. IEEE Computer Society, Los Alamitos (2006)
[22] Woda, M.: Introduction of teaching strategies as a method to increase effectiveness of knowledge acquisition. In: Amman, A.-D.A. (ed.) The 4th International Conference on Information Technology, ICIT 2009, June 3-5, Al-Zaytoonah University of Jordan, Amman, Jordan (2009)
[23] Walkowiak, T., Woda, M.: Komunikacja w systemach e-learningowych. In: Bem i inni, B.J. (red.) Internet 2005, Oficyna Wydaw. PWroc., Wrocław (2005) (in Polish)
[24] Woda, M.: Learning process management in e-learning systems. PhD thesis, Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology (2007)
[25] Woda, M., Kubacki-Gorwecki, K.: Students Learning Styles Classification For e-Education. In: The 5th International Conference on Information Technology, ICIT 2011, Amman, Jordan (2011) (accepted)
[26] Woda, M.: System zdalnego nauczania w ujęciu agentowym. In: Konferencja Nowe media w edukacji. Osiągnięcia pracowników Politechniki Wrocławskiej w zakresie nauczania z wykorzystaniem nowych mediów. Oficyna Wydaw. PWroc, Wrocław (2005) (in Polish)

Incremental Composition of Software Components

W.M. Zuberek[1,2]

[1] Department of Computer Science, Memorial University,
St.John's, NL, Canada A1B 3X5
[2] Department of Applied Informatics, University of Life Sciences,
02-787 Warszawa, Poland
email: wlodek@mun.ca

Abstract. In component-based systems, two interacting components are compatible if all sequences of services requested by one components can be provided by the other component. In the case of several components interacting with a single provider, as is typically the case in client–server computing, the requests from different components can be interleaved and therefore verifying component compatibility must check all possible interleavings of requests from all interacting components. Incremental composition of interacting components eliminates this need for exhaustive combinatorial checking of the interleavings by imposing some restrictions on the interleavings. The paper introduces simple conditions which must be satified by the interacting components for their composition to be incremental and illustrates the concepts using simple examples of interactions.

1 Introduction

Component-base software engineering is one of promising approaches to the development of large-scale software systems [2]. The success of this approach relies, however, on the automated and easily verifiable composition of components and their services [14]. While manual and ad hoc strategies toward component integration have met with some success in the past, such techniques do not lend themselves well to automation. A more formal approach toward the assessment of component compatibility and interoperability is needed. Such a formal approach would permit an automated assessment and would also help promote the reuse of existing software components. It would also significantly enhance the assessment of component substitutability when an existing component is replaced by an improved one, and the replacement is not supposed to affect the functionality of the remaining parts of the system [4].

Components can be considered as the basic functional units and the fundamental data types in architectural design [27]. Components represent high-level software models; they must be generic enough to work in a variety of contexts and in cooperation with other components, but they also must be specific enough to provide easy reuse.

W. Zamojski et al. (Eds.): Dependable Computer Systems, AISC 97, pp. 301–311.
springerlink.com

Primary reasons for component production and deployment are [14]: separability of components from their contexts, independent component development, testing and later reuse, upgrade and replacement in running systems. Component compositionality is often taken for granted. Compositionality, however, is influenced by a number of factors. Component technologies are not entirely independent of particular hardware and operating platforms, programming languages or the specific middleware technology in which they are based. Ideally, the development, quality control, and deployment of software components should be automated similarly to other engineering domains, which deal with the construction of large systems composed of well-understood elements with predictable properties and under acceptable budget and timing constraints [24]. For software engineering, such a situation does not seem to belong to the forseeable future yet.

In this work, two interacting components are considered compatible if any sequence of services requested by one component can be provided by the other. This concept of compatibility can be easily extended to a set of interacting components, then, however, the requests from different components can be interleaved, so any verification of the behavior of the composed system must check all interleavings which can be created by the interacting components. These interleavings can be controlled by specific frameworks for component compositions which can vary from simple syntactic expansions of the GenVoca model [3], to models resembling higher-order programming [5] and dynamic interconnections of distributed processes [22]. Despite significant research efforts, a comprehensive model of software composition is still to come [17].

Incremental composition of interacting components introduces a restriction on the form of interleaving of requests coming from several components, and eliminates the need for exhausting combinatorial checking of the behavior of the composed system. In incremental composition, if a requesting and providing components are compatible, the requesting component can be added to other interacting components without any adverse effect on the behavior of the system. On the other hand, the performance of a system composed in such a way may not be fully used.

Several formal models of component behavior have been proposed in the literature. They include finite automata [9] [10] [28], predicates [29], process calculi [8] [26] and especially labeled Petri nets [1] [12] [16] [19]. Some approaches are built on the concept of subtyping derived from object–oriented programming. They use the interface type to define a subtyping relation between components [7] [21]. Various forms of those types exist, starting with the classical interface type [11] and adding behavioral descriptions such as automata [9]. Related research shows that the resulting approach may be too restrictive for practical applications [29].

Petri nets [23] [25] are formal models of systems which exhibit concurrent activities with constraints on frequency or orderings of these activities. In labeled Petri nets, labels, which represent services, are associated with elements of nets in order to identify interacting components. Well-developed mathematical theory of Petri nets provides a convenient formal foundation for analysis of systems modeled by Petri nets.

This chapter is a continuation of previous work on component compatibility and substitutability [12] [13] [30]. Using the same linguistic specification of component behavior as before [30], the paper introduces icremental component composition and shows that for such a composition, properties of the composed systems can be verified without the exhaustive checking of all possible interleavings of component languages. Simple criteria for incremental composition of components are also given and illustrated by a few examples.

Section 2 recalls the concept of component languages as a characterization of component's behavior. Component languages are used in Section 3 to define component compatibility. Incremental composition is described in Section 4 while Section 5 concludes the chapter.

2 Petri Net Models od Component Behavior

The behavior of a component, at its interface, can be represented by a cyclic labeled Petri net [12] [30]:

$$\mathscr{M}_i = (P_i, T_i, A_i, S_i, m_i, \ell_i, F_i),$$

where P_i and T_i are disjoint sets of places and transitions, respectively, A_i is the set of directed arcs, $A_i \subseteq P_i \times T_i \cup T_i \times P_i$, S_i is an alphabet representing the set of services that are associated with transitions by the labeling function $\ell_i : T_i \to S_i \cup \{\varepsilon\}$ (ε is the "empty" service; it labels transitions which do not represent services), m_i is the initial marking function $m_i : P_i \to \{0, 1, ...\}$, and F_i is the set of final markings (which are used to capture the cyclic nature of sequences of firings).

Sometimes it is convenient to separate net structure $\mathscr{N} = (P, T, A)$ from the initial marking function m.

In order to represent component interactions, the interfaces are divided into *provider* interfaces (or p-interfaces) and *requester* interfaces (or r-interfaces). In the context of a provider interface, a labeled transition can be thought of as a service provided by that component; in the context of a requester interface, a labeled transition is a request for a corresponding service. For example, the label can represent a conventional procedure or method invocation. It is assumed that if the p-interface requires parameters from the r-interface, then the appropriate number and types of parameters are delivered by the r-interface. Similarly, it is assumed that the p-interface provides an appropriate return value, if such a value is required. The equality of symbols representing component services (provided and requested) implies that all such requirements are satisfied.

For unambiguous interactions of requester and provider interfaces, it is required that in each p-interface there is exactly one labeled transition for each provided service:

$$\forall t_i, t_j \in T : \ell(t_i) = \ell(t_j) \neq \varepsilon \Rightarrow t_i = t_j.$$

Moreover, to express the reactive nature of provider components, all provider models are required to be ε–conflict–free, *i.e.*:

$$\forall t \in T\ \forall p \in Inp(t) : Out(p) \neq \{t\} \Rightarrow \ell(t) \neq \varepsilon$$

where $Out(p) = \{t \in T \mid (p,t) \in A\}$; the condition for ε–conflict–freeness could be used in a more relaxed form but this is not discussed here for simplicity of presentation.

Component behavior is determined by the set of all possible sequences of services (required or provided by a component) at a particular interface. Such a set of sequences is called the *interface language*.

Let $\mathscr{F}(\mathscr{M})$ denote the set of firing sequences in $\mathscr{M}$ such that the marking created by each firing sequence belongs to the set of final markings F of $\mathscr{M}$. The interface language $\mathscr{L}(\mathscr{M})$, of a component represented by a labeled Petri net $\mathscr{M}$, is the set of all labeled firing sequences of $\mathscr{M}$:

$$\mathscr{L}(\mathscr{M}) \ = \ \{\ell(\sigma) \mid \sigma \in \mathscr{F}(\mathscr{M})\},$$

where $\ell(t_{i_1} t_{i_2} ... t_{i_k}) = \ell(t_{i_1})\ell(t_{i_2})...\ell(t_{i_k})$.

By using the concept of final markings, interface languages reflect the cyclic behavior of (requester as well as provider) components.

Interface languages defined by Petri nets include regular languages, some context–free and even context–sensitive languages [18]. Therefore, they are significantly more general than languages defined by finite automata [10], but their compatibility verification is also more difficult than in the case of regular languages.

3 Component Compatibility

Interface languages of interacting components can be used to define the compatibility of components; a requester component $\mathscr{M}_r$ is compatible with a provider component $\mathscr{M}_p$ if and only if all sequences of services requested by $\mathscr{M}_r$ can be provided by $\mathscr{M}_p$, i.e., if and only if:

$$\mathscr{L}(\mathscr{M}_r) \subseteq \mathscr{L}(\mathscr{M}_p).$$

Checking the inclusion relation between the requester and provider languages defined by Petri nets $\mathscr{M}_r$ and $\mathscr{M}_p$ can be performed by systematic checking if the services requested by one of the interacting nets can be provided by the other net at each stage of the interaction. In the case of bounded nets, the checking procedure performs a breadth–first traversal of the reachability graph $\mathscr{G}(\mathscr{M}_r)$ verifying that for each transition in $\mathscr{G}(\mathscr{M}_r)$ there is a corresponding transition in $\mathscr{G}(\mathscr{M}_p)$.

3.1 Bounded Models

The following logical function *CheckBounded* can be used for compatibility checking if the requester and provider languages are defined by bounded marked Petri nets

$(\mathcal{N}_r, m_r)$ and $(\mathcal{N}_p, m_p)$, respectively. The function performs exhaustive analysis of the marking spaces of its two argument marked nets checking, at each step, if all service that can be requested by the first argument net are available in the second net. In the pseudocode below, *New* is a sequence (a queue) of pairs of markings to be checked, *head* and *tail* are operations on sequences that return the first element and remaining part of the sequence, respectively, $append(s,a)$ appends an element a to a sequence s, *Analyzed* is the set of markings that have been analyzed, $Enabled(\mathcal{N}, m)$ returns the set of labels of transitions enabled in the net $\mathcal{N}$ by the marking m (including ε if the enabled transitions include transitions without labels), and $next(\mathcal{N}, m, a)$ returns the marking obtained in the net $\mathcal{N}$ from the marking m by firing the transition labeled by x):

```
proc CheckBounded(N_r, m_r, N_p, m_p);
begin
    New := (m_r, m_p);
    Analyzed := {};
    while New ≠ {} do
        (m, n) := head(New);
        New := tail(New);
        if m ∉ Analyzed then
            Analyzed := Analyzed ∪ {m};
            Symbols1 := Enabled(N_r, SkipEps(N_r, m));
            Symbols2 := Enabled(N_p, SkipEps(N_p, n));
            if Symbols1 ∩ Symbols2 = {} then return FALSE fi;
            for each x in Symbols1 do
                if x ∈ Symbols2 then
                    append(New, (next(N_r, m, x), next(N_p, n, x)))
                fi
            od
        fi
    od;
    return TRUE
end;
```

The function *SkipEps(m)* advances the marking function m through all transitions labeled by ε:

```
proc SkipEps(N, m);
begin
    while ε ∈ Enabled(N, m) do m := next(N, m, ε) od;
    return m
end;
```

where the ε parameter of the function *next* refers to any transition enabled by m that is labeled by ε.

The function *CheckBounded* returns TRUE if the language of $(\mathcal{N}_r, m_r)$ is a subset of the language defined by $(\mathcal{N}_p, m_p)$; otherwise FALSE is returned.

Example. Fig.1 shows a simple configuration of two (cyclic) requester components and a single provider of two services named `a` and `b`. In both requester components, the requested services are separated by some "local" operations.

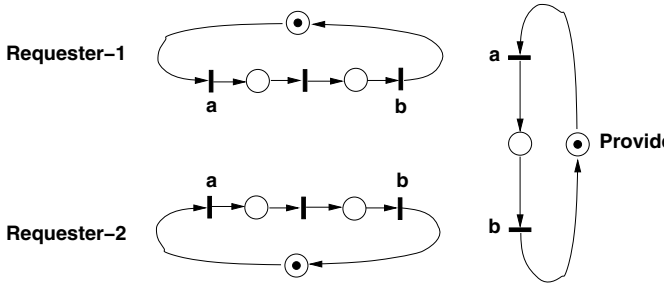


Fig. 1 Two requesters and a single provider.

In this case, the languages of all components are the same, and are sequences of service `a` followed by service `b`. They can be described by a regular exapression `(ab)*`.

For the Requester-1 and Provider nets shown in Fig.1, the steps performed by the function *CheckBounded* can be illustrated in the following table:

m	n	$Symbols1$	$Symbols2$	x	$next(\mathscr{N}_r,m,x)$	$next(\mathscr{N}_p,n,x)$
(1,0,0)	(1,0)	$\{a\}$	$\{a\}$	a	(0,1,0)	(0,1)
(0,1,0)	(0,1)	$\{b\}$	$\{b\}$	b	(1,0,0)	(1,0)

Since in each case, the (only) symbol of $Symbols1$ is also an element of $Symbols2$, the returned result of checking is TRUE.

3.2 Unbounbded Models

For the unbounded case, compatibility checking must include checking the unboundedness condition (a marked net $(\mathscr{N},m_0)$ is unbounded if there exist markings m' and m'' reachable from m_0 such that m'' is reachable from m' and m'' is componentwise greater or equal to m'). This condition is checked for the requester as well as for the provider nets by combining these two markings together. More specifically, for each analyzed pair of markings (m,n), an additional check is performed if the set *Analyzed* contains a pair of markings, which is componentwise smaller than (m,n) and from which (m,n) is reachable; if the set *Analyzed* contains such a pair, analysis of (m,n) is discontinued. This additional check is performed by a logical function $Reachable((m,n),Analyzed)$:

```
proc CheckUnbounded(N_r, m_r, N_p, m_p);
begin
    New := (m_r, m_p);
    Analyzed := {};
    while New ≠ {} do
        (m,n) := head(New);
        New := tail(New);
        if (m,n) ∉ Analyzed then
            Analyzed := Analyzed ∪ {(m,n)};
            Symbols1 := Enabled(N_r, SkipEps(N_r, m);
            Symbols2 := Enabled(N_p, SkipEps(N_p, n);
            if Symbols1 ∩ Symbols2 = {} then return FALSE fi;
            if not Reachable((m,n).Analyzed) then
                for each x in Symbols1 do
                    if x ∈ Symbols2 then
                        append(New, (next(N_1, m, x), next(N_2, n, x))
                    fi
                od
            fi
        fi
    od;
    return TRUE
end;
```

Example. Fig.2 shows a modified model of a provider which still requires that each operation `b` is preceded by an operation `a`, but which also allows several operations `a` to be performed before any of the corresponding `b` operations is requested (which is not allowed in model shown in Fig.1). This provider net is unbounded.

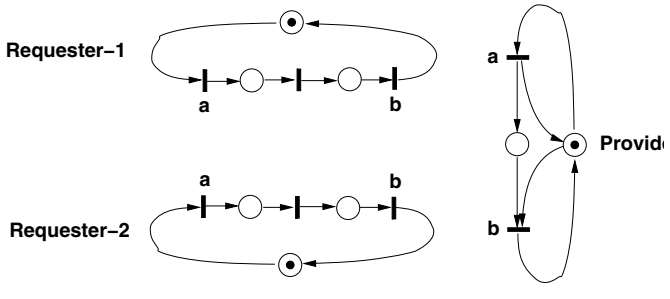


Fig. 2 Two requesters with a modified provider.

Checking the compatibility of Requester-1 and Provider in Fig.2, performed by *CheckUnboundedOne*, can be illustrated by the following table:

m	n	$Symbols1$	$Symbols2$	x	$next(\mathscr{N}_r,m,x)$	$next(\mathscr{N}_p,n,x)$
(1,0,0)	(1,0)	$\{a\}$	$\{a\}$	a	(0,1,0)	(1,1)
(0,1,0)	(1,1)	$\{b\}$	$\{a,b\}$	b	(1,0,0)	(1,0)

Again, since for each case the (single) element of *Symbols1* is also an element of *Symbols2*, the result of checking is TRUE.

4 Incremental Composition

Incremental composition of interacting components takes advantage of the cyclic nature of component behavior, and allows the interleaving at the level of cycles. So, taking this behavioral cyclicity into account, the condition of compatibility of interacting components can be rewritten as:

$$\mathscr{L}_r^* \subseteq \mathscr{L}_p^*$$

where $\mathscr{L}_r$ is the single-cycle language of the requester component and $\mathscr{L}_p$ is the single-cycle language of the provider component. These single-cycle languages can be just sequences of services (requested or provided) but normally they are more sophisticated and can even be infinite.

The above compatibility condition can be simplified to:

$$\mathscr{L}_r \subseteq \mathscr{L}_p.$$

For the case of k requester components interacting with a single provider, for incremental composition the combined language of requesters becomes:

$$(\mathscr{L}_{r1} \cup \mathscr{L}_{r2} \cup \cdots \cup \mathscr{L}_{rk})^*$$

and then the simplified compatibility condition is:

$$\mathscr{L}_{r1} \cup \mathscr{L}_{r2} \cup \cdots \cup \mathscr{L}_{rk} \subseteq \mathscr{L}_p$$

which is equivalent to

$$\mathscr{L}_{r1} \subseteq \mathscr{L}_p \wedge \mathscr{L}_{r2} \subseteq \mathscr{L}_p \wedge \cdots \wedge \mathscr{L}_{rk} \subseteq \mathscr{L}_p$$

so, instead of checking the compatibility of interleaved requests, it is sufficient to check if each requester component is compatible with the provider. Consequently, the incremental composition eliminates the need for exhaustive combinatorial checking of the behavior of the composed system.

It can be observed that a straightforward criterion for incremental composition is that the set *First(L_r)* of leading symbols of the single-cycle languge $\mathscr{L}_r$ is disjoint with the set *Follow(L_p)* of non-leading symbols of the single-cycle language $\mathscr{L}_p$:

$$First(\mathscr{L}_r) \cap Follow(\mathscr{L}_p) \;=\; \{\}$$

where (S is the set of services required and provided by the components):

$$First(\mathscr{L}) = \{a \in S \mid \exists x \in S^* : ax \in \mathscr{L}\},$$
$$Follow(\mathscr{L}) = \{a \in S \mid \exists x \in S^+, y \in S^* : xay \in \mathscr{L}\}.$$

Example. The languages of Requester-1, Requester-2 and Provider shown in Fig.1 are $(\mathtt{ab})*$, their single-cycle languages are $(\mathtt{ab})$, so $First(\mathscr{L}_r) = \{\mathtt{a}\}$, $Follow(\mathscr{L}_p) = \{\mathtt{b}\}$, and $First(\mathscr{L}_r) \cap Follow(\mathscr{L}_p) = \{\}$, so the composition is incremental.

For Fig.2, the languages of Requester-1 and Requester-2 are also $(\mathtt{ab})*$, but the language of Provider is nonregular. In this case, $First(\mathscr{L}_r) = \{\mathtt{a}\}$, $Follow(\mathscr{L}_p) = \{\mathtt{a}, \mathtt{b}\}$, and $First(\mathscr{L}_r) \cap Follow(\mathscr{L}_p) \neq \{\}$, so the composition of these models is not incremental and requires a more detailed verification.

5 Concluding Remarks

Incremental composition eliminates the exhaustive verification of the behavior of the composed system by restricting the behavior of interacting components. A simple criterion can be used to check if the interacting components satisfy the requirement of incremental composition.

A different approach, called component adaptation is proposed in [6]. Its main idea is to identify mismatches of interacting components and to generate (on the basis of formal specification of components) component adaptors which eliminate the identified mismatches. The formal foundation for such an approach is provided in [28].

A similar approach is proposed in [20].

Different languages, tools, and environments that support some degree of component–oriented development, support some kinds of components and component composition, but no common model exists. Therefore it is difficult to compare different approaches in a unifor way and is difficult to reason about inteoperability between languages and platforms. More research in this area is expected.

On the other hand, an interesting (and challenging) task that needs to be addressed is the derivation of Petri net behavioral models of components. The derivation should be automated using either component formal specifications or component implementations.

Acknowledgement. The Natural Sciences and Engineering Research Council of Canada partially supported this research through grant RGPIN-8222. Helpful remarks of three anonymous reviewers are gratefully acknowledged.

References

1. Aalst van der, W.M.P., Hee van, K.M., Torn van der, R.A.: Component-based software architecture: a framework based on inheritance of behavior. Science of Computer Programming 42(2-3), 129–171 (2002)

2. Attiogbé, J.C., André, P., Ardourel, G.: Checking component composability. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, pp. 18–33. Springer, Heidelberg (2006)
3. Batiry, D., Singhal, V., Thosmas, J., Dasari, S., Geract, B., Sirkin, M.: The Gen Voca model of software system generators. IEEE Software 11(5), 89–94 (1994)
4. Belguidoum, M., Dagnat, F.: Formalization of component substitutability. Electronic Notes in Theoretical Computer Science 215, 75–92 (2008)
5. Bracha, G., Cook, W.: Mixin-based inheritance. In: Proc. Joint ACM Conf. on Object-Oriented Pogramming, Systems, Languages and Applications and the u European Conf. on Object-Oriented Programming, pp. 303–311 (1990)
6. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptations. The Journal of Systems and Software 74(1), 45–54 (2005)
7. Brada, P., Valenta, L.: Practical verification of component substitutability using subtype relation. In: Proc. Int. Conf. on Software Engineering and Advanced Applications (SEAA 2006), pp. 38–45 (2006)
8. Canal, C., Pimentel, E., Troya, J.M.: Compatibility and inheritance in software architectures. Science of Computer Programming 41(2), 105–138 (2001)
9. Cerna, I., Varekove, P., Zimmerova, B.: Component substitutability via equivalencies of component-interaction automata. In: Proc. Int. Workshop on Formal Aspects of Component Software (FACS 2006), pp. 115–130 (2006)
10. Chaki, S., Clarke, S.M., Groce, A., Jha, S., Veith, H.: Modular verification of software components in C. IEEE Trans. on Software Engineering 30(6), 388–402 (2004)
11. Costa Seco, J., Caires, L.: A basic model of typed components. In: Proc. 14-th European Conf. on Object-Oriented Programming, London, UK, pp. 108–128 (2000)
12. Craig, D.C., Zuberek, W.M.: Compatibility of software components – modeling and verification. In: Proc. Int. Conf. on Dependability of Computer Systems, Szklarska Poreba, Poland, pp. 11–18 (2006)
13. Craig, D.C., Zuberek, W.M.: Petri nets in modeling component behavior and verifying component compatibility. In: Proc. Int. Workshop on Petri Nets and Software Engineering, Siedlce, Poland, pp. 160–174 (2007)
14. Crnkovic, I., Schmidt, H.W., Stafford, J., Wallnau, K.: Automated component-based software engineering. The Journal of Systems and Software, vol 74(1), 1–3 (2005)
15. Garlan, D.: Formal modeling and analysis of software architecture: Components, connectors, and events. In: Bernardo, M., Inverardi, P. (eds.) SFM 2003. LNCS, vol. 2804, pp. 1–24. Springer, Heidelberg (2003)
16. Hameirlain, N.: Flexible behavioral comatibility and substitutability for component protocols: a formal specification. In: Proc. 5-th Int. Conf. on Software Engineering and Formal Methods, London, England, pp. 391–400 (2007)
17. Henrio, L., Kammüller, F., Khan, M.U.: A framework for reasoning on component composition. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) FMCO 2009. LNCS, vol. 6286, pp. 1–20. Springer, Heidelberg (2010)
18. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computations, 2nd edn. Addison-Wesley, Reading (2001)
19. Karlsson, D., Eles, P., Peng, Z.: Formal verification on a component-based reuse methodology. In: Proc. 15-th Int. Symp. on System Synthesis, Kyoto, Japan, pp. 156–161 (2002)
20. Leicher, A., Busse, S., Süß, J.G.: Analysis of compositional conflicts in component-based systems. In: Gschwind, T., Aßmann, U., Wang, J. (eds.) SC 2005. LNCS, vol. 3628, pp. 67–82. Springer, Heidelberg (2005)
21. Liskov, B., Wing, J.: A behavioral notion of subtyping. ACM Trans. on Programming Languages and Systems 19(6), 1811–1841 (1994)

22. Magee, J., Dulay, N., Kramer, J.: Specifying distributed software architectures. In: Botella, P., Schäfer, W. (eds.) ESEC 1995, Sitges, Spain, LNCS, vol. 989, pp. 137–153 (1995)
23. Murata, T.: Petri nets: properties, analysis, and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
24. Nierstrasz, O., Meijler, T.: Research directions on software composition. ACM Computing Surveys 27(2), 262–264 (1995)
25. Reisig, W.: Petri nets – an introduction (EATCS Monographs on Theoretical Computer Science 4). Springer, Heidelberg (1985)
26. Südholt, M.: A model of components with non-regular protocols. In: Gschwind, T., Aßmann, U., Wang, J. (eds.) SC 2005. LNCS, vol. 3628, pp. 99–113. Springer, Heidelberg (2005)
27. Szyperski, C.: Component software: beyond object-oriented programming, 2nd edn. Addison–Wesley Professional, London (2002)
28. Ywllin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. on Programming Languages and Systems 19(2), 292–333 (1997)
29. Zaremski, A.M., Wang, J.M.: Specification matching of software components. ACM Trans. on Software Engineering and Methodology 6(4), 333–369 (1997)
30. Zuberek, W.M.: Checking compatibility and substitutability of software components. In: Models and Methodology of System Dependability, ch.14, pp. 175–186. Oficyna Wydawnicza Politechniki Wroclawskiej, Wroclaw (2010)

Author Index