THESIS ON INFORMATICS AND SYSTEM ENGINEERING C119

Software Technology for Cyber Security Simulations

ANDRES OJAMAA

TUT PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
Institute of Cybernetics

Dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer Science on October 25, 2016

Supervisors: Enn Tõugu, D. Sc.
Leading Researcher, Institute of Cybernetics
Tallinn University of Technology, Tallinn, Estonia

Jaan Penjam, PhD
Senior Researcher, Institute of Cybernetics
Tallinn University of Technology, Tallinn, Estonia

Opponents: Margus Veanes, PhD
Research in Software Engineering (RiSE) Group
Microsoft Research, Redmond, USA

Christian Czosseck, PhD
Head Laboratory at CERT Bw, Germany

Defence of the thesis: December 15, 2016

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/ Andres Ojamaa /

ISSN 1406-4731
ISBN 978-9949-83-051-0 (publication)
ISBN 978-9949-83-052-7 (PDF)

INFORMAATIKA JA SÜSTEEMITEHNIKA C119

Tarkvaratehnika küberturbe simulatsioonide jaoks

ANDRES OJAMAA

TTÜ
KIRJASTUS

Contents

List of Figures

List of Publications

The work of this thesis is based on the following publications:

I Kotkas, Vahur; **Ojamaa, Andres**; Grigorenko, Pavel; Maigre, Riina; Harf, Mait; Tyugu, Enn (2011). CoCoViLa as a Multifunctional Simulation Platform. Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques: 21–25 March 2011, Barcelona, Spain, SIMUTools 2011. Brussels: ICST, 195–205.

II Klein, Gabriel; **Ojamaa, Andres**; Grigorenko, Pavel; Jahnke, Marko; Tyugu, Enn (2010). Enhancing Response Selection in Impact Estimation Approaches. Concepts and Implementations for Innovative Military Communications and Information Technologies. Ed. Amanowicz, Marek. Warsaw: Military University of Technology, 277–286.

III **Ojamaa, Andres** (2009). Hybrid Simulation of Large Networks. Proceedings of the 2009 International Conference on Modeling, Simulation & Visualization Methods, MSV 2009. Ed. Arabnia, Hamid R.; Deligiannidis, Leonidas. Las Vegas: CSREA Press, 219–225.

IV **Ojamaa, Andres**; Tyugu, Enn (2007). Rich Components of Extendable Simulation Platform. Proceedings of the 2007 International Conference on Modeling, Simulation & Visualization Methods, MSV 2007: June 25–28 2007, Las Vegas Nevada, USA. Ed. Arabnia, Hamid R. Las Vegas: CSREA Press, 121–127.

V **Ojamaa, Andres**; Haav, Hele-Mai; Penjam, Jaan (2015). Semi-Automated Generation of DSL Meta Models from Formal Domain Ontologies. Model and Data Engineering: 5th International Conference, MEDI 2015, Rhodes, Greece, September 26–28, 2015, Proceedings. Ed. Bellatreche, Ladjel; Manolopoulos, Yannis. Springer, 3–15. (Lecture Notes in Computer Science; 9344).

VI Haav, Hele-Mai; **Ojamaa, Andres**; Grigorenko, Pavel; Kotkas, Vahur (2015). Ontology-Based Integration of Software Artefacts for DSL Development. On the Move to Meaningful Internet Systems: OTM 2015 Workshops: Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, EI2N, FBM, INBAST, ISDE, META4eS, and MSC 2015, Rhodes, Greece, October 26–30, 2015, Proceedings. Ed. Ciuciu, I. et al. Cham: Springer, 309–318. (Lecture Notes in Computer Science; 9416).

VII Haav, Hele-Mai; **Ojamaa, Andres** (2016). Semi-Automated Integration of Domain Ontologies to DSL Meta-Models. *International Journal of Intelligent Information and Database Systems*. [accepted]

VIII Kivimaa, Jyri; **Ojamaa, Andres**; Tyugu, Enn (2009). Graded Security Expert System. Critical Information Infrastructures Security: Third International Workshop, CRITIS 2008, Rome, Italy, October 13–15, 2008, Revised Papers. Ed. Setola, Roberto; Geretshuber, Stefan. Berlin: Springer, 279–286. (Lecture Notes in Computer Science; 5508).

IX **Ojamaa, Andres**; Tyugu, Enn; Kivimaa, Jyri (2008). Pareto-Optimal Situation Analysis for Selection of Security Measures. MILCOM 08: Assuring Mission Success: Unclassified Proceedings, November 17–19 San Diego. 3224–3230.

X Kivimaa, Jyri; **Ojamaa, Andres**; Tyugu, Enn (2009). Managing Evolving Security Situations. MILCOM 2009: Unclassified Proceedings, October 18–21, 2009, Boston, MA. Piscataway, NJ: IEEE, 1–7.

XI **Ojamaa, Andres**; Tyugu, Enn (2016). Enterprise Security Analysis and Training Experience. Critical Information Infrastructures Security: 9th International Conference, CRITIS 2014, Limassol, Cyprus, October 13–15, 2014, Revised Selected Papers. Ed. Panayiotou, C.G.; Ellinas, G.; Kyriakides, E.; Polycarpou, M.M. Cham: Springer, 200–208. (Lecture Notes in Computer Science; 8985).

Author's contributions to the publications

The author's contributions to the publications are summarized here. The roman numerals in front of the list items correspond to those in the list of publications.

I Introducing and prototyping the concept of *rich components* (related to Publication IV), proposing and implementing a hybrid network simulation technique (Publication III), CoCoViLa software development, design and implementation of example applications (e.g., Publication II), writing parts of the paper, conference presentation.

II An implementation of the GrADAR method as a CoCoViLa package including an optimizer component, writing parts of the paper.

III Proposing a hybrid approach to modeling and simulating large networks, developing a prototype, writing the paper, conference presentation.

IV Proposing a new type of software component, developing a prototype, writing parts of the paper, conference presentation.

V Proposing the idea to use OWL ontologies for generating DSL metamodels, performing experiments and implementing a prototype, writing parts of the draft, conference presentation.

VI Proposing the architecture for integrating software artefacts, performing experiments and implementing a prototype.

VII Proposing the idea to use OWL ontologies for generating DSL metamodels, developing the architecture for integrating software artefacts, performing experiments and implementing a prototype, writing parts of the draft.

VIII Formalizing the graded security model originally conceived on the conceptual level by Dr Kivimaa, designing and implementing a prototype expert system for supporting security investment optimization based on the graded security model, developing visualization tools, finding and implementing an efficient optimization algorithm, writing parts of the paper, conference presentation.

IX Developing an approach for designing optimal security solutions based on the graded security model provided by Dr Kivimaa, developing software and algorithms, writing parts of the paper, live demonstration at the conference presentation.

X Extending the existing model and expert system to cover more general cases and to support developing long-term security investment plans, developing algorithms and software, writing parts of the paper, conference presentation.

XI Using and supporting the tools for teaching Cyber Security Master's students at Tallinn University of Technology, software development, writing parts of the paper, conference presentation.

1 Introduction

Cyber security is a multi-disciplinary field. The view that cyber security is just a technical issue might have been common a decade or two ago [80], but this view has become less common and less true [81]. Securing IT systems of a modern organization requires effort from different roles and cooperation between all departments.

From the technical perspective, there is a general direction for IT systems to become more connected, hence open to the external environment. The factors driving this include the need to get information online, deploy software updates and perform diagnostics remotely [51]. This trend applies also to control systems that may have the potential to cause physical damage in the real world. Therefore, cyber security is not just about assuring information confidentiality, integrity, availability and protecting IT systems from abuse, but is also essential for maintaining safety in physical space.

Cyber security experts need to analyze technical and non-technical aspects of security, integrate data from multiple sources and communicate across the organization with people in different positions. This work can be facilitated by using the support of flexible software tools for analyzing security situations, solving operational security issues, optimizing available resources and presenting the results to decision makers in an efficient way.

Simulation is a universal technique for studying complex systems. In this work the focus is on simulations performed on computers. Computer simulations require the development of formal models of systems and allow to conduct simulation experiments. This process can lead to a better understanding of the behaviour of the simulated system. Simulation allows to experiment with different *what-if* scenarios and to gain new knowledge. Getting a good understanding of a system is often a very important first step in securing it.

Computer simulation is an established and a valuable practice in conventional engineering disciplines. However, it seems the full potential of computer simulation is not used in the cyber security field. An underlying assumption of this work is that better software tools and technologies could help to increase the use of computer simulation in the cyber security field leading to more efficient usage of resources, better communication between stakeholders and, finally, a higher level of cyber security.

The motivation for the work comes mainly from the following questions:

- What kind of simulation tools could help to counter increasing threats in the cyber space?
- How could existing software engineering technologies be improved in order to provide more value to cyber security practitioners?

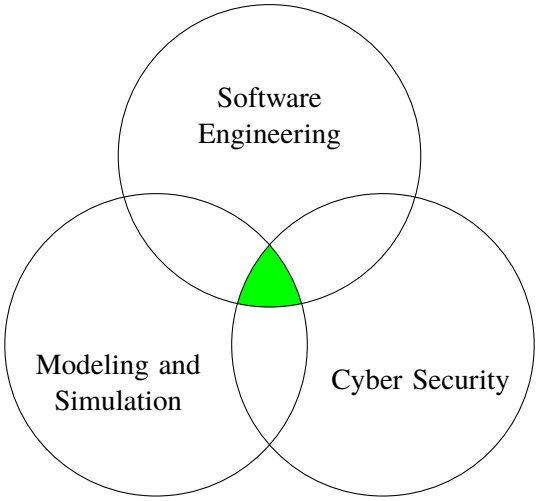


Figure 1.1: Focus area of the dissertation

- Is there anything special in modeling and simulation in the cyber security field compared to other more conventional fields?

Hence, the field of this thesis is related to the intersection of three different domains—software engineering, modeling and simulation, and cyber security. The area of this topic is illustrated in Figure 1.1. To provide a broader context for this work, the three domains are briefly discussed in the following sections.

1.1 Software Engineering

This section discusses general aspects of software engineering, focusing on a particular approach to software development—model based software engineering. The key issues of interest in the context of this work are software development productivity, flexibility of software products and reuse of software assets in the context of cyber security simulation.

The term *software engineering* was coined in a 1967 talk [53] by Oettinger. In this talk the essence of computer science and its relations to mathematics, logic and classical engineering were discussed. Also, it was argued that the broad span of software development from the most abstract mathematics to the dirtiest of unconventional engineering makes software development difficult and interesting. It was recognized that the complexity and cost of software development is a major engineering challenge.

The report [52] of the first software engineering conference in 1968 highlights discussions raising concerns about inadequate productivity of software development. Many of these discussions are still relevant. The relevant subjects include:

- "the problems of achieving sufficient reliability in the data systems which are becoming increasingly integrated into the central activities of modern society";
- "the difficulties of meeting schedules and specifications on large software projects";
- "the education of software (or data systems) engineers";
- the lack of best practices for software development.

An influential essay titled *Essence and Accidents of Software Engineering* [9] by Brooks explains that the complexity of software is an *essential* property, not an *accidental* one, and essential complexity cannot be removed. Accidental complexity can be reduced by using better tools, processes, methods, but orders of magnitude productivity gains should not be expected from this.

The 2015 Standish Group CHAOS report [36] summarizes the results of the study of 50 thousand software projects[1]. According to the report, only 29% of the projects were successful, 52% were challenged in some way and 19% failed completely. Although the methodology of the study has been criticised [4,20] there still seems to exist a major engineering challenge.

The discipline of software engineering originates from the need to address poor quality of software, get projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification [64]. In this work the following definition of software engineering is used [23]:

> Software engineering is the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.

Better productivity of software development can arise from well-suited abstractions, better tools and processes, higher level of automation, and more efficient reuse of existing artifacts [82]. The need to measure and improve software development productivity has stimulated research in this field [62].

Model Based Software Engineering (MBSE) is a systematic approach to software development where software models are used in order to improve the productivity and quality of software creation. In MBSE, software models, besides being a part of documentation, become essential artefacts in the process of software development. As a consequence, this forces the models to be formalized, complete and precise. There are a number of ways for practicing MBSE which differ in various specific aspects. By and large, the goals in all cases include automation of some development steps, higher flexibility, i.e., existing artefacts can be adapted

[1] `http://www.infoq.com/articles/standish-chaos-2015`

for new purposes relatively easily, and working at an appropriate level of abstraction, hence increasing productivity. A number of frameworks, platforms and tools for supporting MBSE approaches have been developed, for example, MetaEdit+, Atom3, Eclipse EMF, OMG MDA, IBM Rational Rhapsody. It has been suggested that model engineering can be considered a new sub-discipline to software engineering [25]. Several studies have analyzed the advantages and disadvantages of model based approaches [7, 67, 68]. Model based software engineering has been found to be a useful methodology for handling the complexity in developing models for simulation studies of complex systems [21].

Models can also represent other models. Models that describe a set of models are called *meta-models*. A model is an instance of a corresponding meta-model. Meta-models specify valid building blocks and the structure of models. In the context of MBSE, a meta-model is a model of a modeling language including its abstract syntax and static semantics used for describing domain concepts and relations that constitute domains models. Explicit formal meta-models are an important part of MBSE as they enable automation of software development (e.g., program code generation).

A model based software engineering technology developed at the Institute of Cybernetics at Tallinn University of Technology has been demonstrated to be suitable for engineering applications [33,35]. This technology is referred to as *CoCoViLa technology*. An overview of the platform supporting this technology is given in Chapter 2. One of the goals of this work is to verify the applicability of CoCoViLa technology to the cyber security domain and to suggest, based on practical experiments, improvements to the technology.

1.2 Modeling and Simulation

A *simulation* is the imitation of the operation of a real-world process or a system over time [6]. A *model* is an abstract and formal representation of a process or system that includes the attributes and properties of the modeled object which are relevant to the current study. In general, the purposes of modeling and simulation may include: to explore possible options; communicate knowledge to others; predict; explain existing data [79].

Simulations are used to study complex systems for gaining a better understanding of the behavior of the systems and to analyze what-if scenarios. Simulations are also often useful as educational tools. Simulated situations are helpful to prepare for different scenarios. Often, a simulation can be the only way to experiment a system, especially when the system is very expensive or dangerous.

There are, however, cases when simulation is not an appropriate approach. For example, there is no reason to spend resources on simulation when a good analytical solution exists. Other potential risks and problems include: the garbage in, garbage

out problem; high development cost; lack of input data; inability to perform proper validation; interoperability issues.

"Most of the well-known modeling and simulation (M&S) methodologies state the importance of conceptual modeling in simulation studies, and they suggest the use of conceptual models during the simulation model development process. However, only a limited number of methodologies refers to how to move from a conceptual model to an executable simulation model. Besides, existing M&S methodologies do not typically provide a formal method for model transformations between the models in different stages of the development process. Hence, in the current M&S practice, model continuity is usually not fulfilled." [12]

1.3 Cyber Security

Cyber security is a broad field that includes technical, economic, social and military aspects. The general goal of cyber security is to protect the availability, confidentiality and integrity of information (the classical CIA model of information security) as well as ensure the availability and correct operation of computer systems and the facilities controlled by these computer systems.

There are several definitions of "cyber security" and the relation of it to "information security". Unfortunately, a universally accepted and consistently used terminology does not exist in this field. In this work the concept "cyber security" is used in the meaning as discussed in a paper by von Solms [78]. It is important to note that although "cyber security" and "information security" have a substantial overlap, the concepts are not totally analogous. For instance, cyber security also covers incidents where IT infrastructure is abused but information security is not compromised. In the context of this work the term "information technology (IT) security" is a synonym of "cyber security".

The term "science of cyber security" has appeared in the literature. It has been defined [43] as "the study of relations—preferably expressed as theoretically-grounded models—between attributes, structures and dynamics of: violations of cyber security policy; the network of computing devices under attack; the defenders' tools and techniques; and the attackers' tools and techniques where malicious software plays the central role."

In general, security is hard to model and often impossible to prove. On the one hand, for any non-trivial real world system the absence of security issues cannot be shown. The attacker, on the other hand, usually needs to find and exploit just a small number of security holes to succeed. This property implies a strong asymmetry between defensive and offensive sides.

Situation awareness is essential in cyber security operations. It depends on a reliable perception of the environment and comprehension of its semantic structures. In this respect, cyber space presents a unique challenge to the situation awareness

of users and analysts, since it is a unique combination of human and machine elements, whose complex interactions occur in a global communication network [56].

The processes in the cyber space are very fast compared to human reaction time, invisible to the naked eye and involve large amounts of data. However, the technical side is just one part of the complete picture. As the cyber space has become an universal medium in the global economy, social aspects and human behavior also have to be taken into consideration.

For responding to cyber security incidents time is critical. Studies have concluded [13, 69] that the success rate of an intruder rises with the time he or she can work undisturbed: a skilled attacker can perform an intrusion with an 80% success rate if given 10 hours time before any response is launched.

Simulations in cyber security can be approached from different viewpoints. In the context of this work, the following three characteristics of cyber security simulations are of interest.

First, *what are the properties of cyber security that make simulation as a method usually well applicable to the problems in this field.* A prominent reason is that our normal everyday life depends heavily on IT systems. Therefore, on the one hand, it would be impossible to take these systems offline for experiments and testing. On the other hand, conducting experiments on an important live system can be too risky. It would also be impractical to replicate such systems in a test lab due to the cost of hardware, labour and maintenance. Hence, simulation mostly in software might be the only feasible approach to study these systems. Experimenting and testing, however, is a vital requirement to develop and deploy new technologies, protocols and defense measures in a quickly evolving cyber space. Another reason to use simulations is, as discussed above, the fast and invisible nature of the processes in computer systems. Simulation-based automated decision support systems can help people in making correct decisions rapidly in critical situations. Visualization and data analysis, which are parts of any simulation study, are also invaluable in monitoring, understanding and predicting the behaviour of complex computer systems.

Second, *what are reasons why computer systems can be easier to model and simulate compared to, for example, some biological systems.* One crucial difference is that the cyber space is entirely artificial and human made, also often quite deterministic and understandable in detail. As the cyber space is mostly digital and discrete, it can modeled using logic. This can simplify the task of modeling by a large factor. But still, for example, the modeling of some network protocols can be far from trivial because of the built-in adaptiveness and the closed-loop design of these network protocols. Nevertheless, verifying hypotheses and experimenting with different configurations can be again relatively easy in the digital world, because most of the behaviour of the systems is determined by software which can be modified on the fly without expensive equipment.

Third, *what are the properties that make it hard to simulate cyber security.* To begin with, there are some fundamental reasons that cause cyber security to be

very hard to model accurately. One of the reasons being that the attacker can be intelligent and possess information unknown to the defender. Attackers can also be rational, seeking to maximize their financial benefits, or socially motivated and behaving irrationally. The situation becomes even more interesting when human adversaries with different motivations are combined with intelligent agent-based malware [72]. Also, based on a survey taken in 2006, it has been estimated that approximately 60% of security breaches were attributed to human error by security managers [8].

Another fundamental aspect of security is that the overall success of an attack is determined by the weakest link in the chain. This weakest link can be a minor technical detail, maybe just one character in a program source code. It is not feasible to model large-scale systems with that amount of details. Any simplification to the model can, in principle, leave out crucial information. Besides these fundamental problems, accurately simulating large computer systems tends to require processing huge amounts of data which is computationally expensive. Unlike simulations of other domains, in the case of a security incident it is not possible to wait for hours or even days for the cyber security simulation to finish. By that time the results may be out of date to the point of being completely unusable or misleading.

Having discussed the characteristics of cyber security simulation problems, it can be concluded that while these problems share many properties with any other simulation problem, the particular combination of characteristics sets certain requirements to the tools and methodologies used in this field. That leads us to the motivation and goals of this work stated in the following section.

1.4 Motivation and Objectives

As the real world and cyber space have become interdependent, the importance of security concerns has also increased dramatically—a security breach in the cyber space may have some serious implications in the real world (e.g., the KWC water utility company incident [77]), and vice versa. It has been reported [16] that 90% of large organisations and 74% of small businesses had an IT security breach in 2015 while 59% of respondents expect there will be more security incidents in the next year. Hence, it is necessary to develop methodologies, techniques and tools for advancing the state of the art in cyber security to keep up with the increased level of risk.

This work is about using computer simulations in the cyber security domain for helping to improve the overall safety of computer systems, individuals, organizations and states. As this field is relatively young and immature, there is a lack of proven methodologies and smart simulation tools. Smart software tools are programs that take over some responsibility usually performed by intelligent users, for example, specifying solution steps. This thesis is an attempt to broaden the availability of such methodologies and tools.

Despite the existence of large well-established simulation software packages and powerful personal computers, accurate and efficient cyber security simulations still remain a challenge. Therefore, new advanced techniques and tools are needed which would accommodate the diverse requirements demanded by simulations in the evolving cyber security domain. A suitable methodology combined with software tools would conform to the following points.

- It should be possible to model and simulate every important aspect of a complex cyber security related system.
- The tools have to be usable for domain experts without the need to write program code.
- Multi-scale multifunctional simulations should be supported based on the technology currently available.
- The tools should be open and flexible to be able to evolve together with the tasks.
- Supporting educational exercises as well as solving large scale real world problems is equally important.

Multi-functional and multi-paradigm modeling and simulation presents technical challenges—how to integrate different domain models, technologies and tools. This can be solved with tools and technologies. In addition, there is a more fundamental issue of model validation. A model that integrates components built on contradicting assumptions cannot give good results. This is a really hard problem and it is unlikely an easy universal solution can be found.

The goal of the work is to develop smart cyber security simulation tools. This includes methods, technology and freely available software tools for cyber security simulation that will be applicable to wide set of problems and will be economical and time-efficient, while still providing the required precision.

1.5 Contributions

This work resulted in three main contributions which are summarized here.

- First, a novel program synthesis based methodology for performing cyber security simulations was proposed. This methodology was implemented in a multifunctional simulation platform. This platform supports model based software engineering, adds flexibility and improves productivity by employing automatic synthesis of programs, and provides the infrastructure for performing complex multi-scale hybrid simulations. Another distinguishing feature of this platform is the support of a new type of software building blocks—rich components.

- Second, extensions to an existing model based software engineering technology were proposed. These extension introduce OWL ontologies into the domain engineering process facilitating knowledge and software reuse, improving consistency. Furthermore, the CoCoViLa modeling and simulation environment's specification capabilities were extended with the support of OWL DL inference and production rules that increase expressiveness and add flexibility.

- Third, testing the proposed approach and simulation platform for solving practical problems. Three packages, including sophisticated optimization methods, for solving different simulation tasks were developed as a part of this work. One of these applications, an expert system for cyber security cost optimization, is presented in detail in this dissertation.

These results have been used for several purposes. A graduate-level simulation course was prepared and given at Tallinn University of Technology in Autumn 2008 to the Master's students of the Cyber Defense module. The students solved practical exercises using the software developed in the scope of this work. A similar but more extensive simulation course was given in the Autumn 2012 semester.

The software and methods have also been used at the Cooperative Cyber Defence Centre of Excellence (CCD COE), as the Centre was one of the initiators and supporters of this work. Furthermore, the results have been applied in pilot projects in banks: the Estonian branches of SEB Bank and Swedbank [41].

The outcomes of this work has been a basis for further academic and research projects. For example, several conference papers [3, 40, 42] have been published describing new work based on the original contributions of this thesis.

Moreover, the outcomes of this work—including the formalization of the initial graded security model originally conceived on the conceptual level by Dr Kivimaa and later extended as joint work; graded security expert system design and implementation; applying the model based software engineering approach to solving the graded security cost optimization problem, the implementation of an efficient optimizer—have provided groundwork for later developments and extensions of the graded security model, graded security expert system and simulation experiments used in Dr Kivimaa's PhD thesis [41] and in several cyber security related Master's theses (e.g., [2, 39]) supervised by Dr Kivimaa. The author of this work was not involved in the development of the *graph-based graded security model* that forms a core part of Dr Kivimaa's dissertation [41].

As a result of the work presented in this dissertation it was shown that:

- A general purpose model based software engineering tool CoCoViLa can be used for several different simulation problems in the cyber security simulations.

- Model based approach to software engineering combined with domain specific visual languages is well suited for the use by cyber security domain experts.
- Component oriented technology with automatic synthesis of programs used in CoCoViLa provides flexibility and supports the reuse of program components and domain knowledge.

1.6 Organization of the Dissertation

The dissertation is based on the author's publications and gives an overview of the work done during his doctoral studies. The rest of the thesis is structured as follows. First, the CoCoViLa simulation platform and the software engineering technology is introduced in Chapter 2 (Publications I, II, III and IV). The chapter also briefly discusses some application examples. Chapter 3 is focused on the ontology based software development technology. The extensions of the technology developed by the author are described there (Publications V, VI and VII). Next, in Chapter 4 an example application is presented demonstrating the applicability of the technology in the cyber security domain (Publications VIII, IX, X and XI). Finally, in Chapter 5, the results are discussed and conclusions are drawn together with suggestions for future research.

2 Software Platform for Simulation

This chapter presents the CoCoViLa platform with the focus on simulation applications in the cyber security domain. CoCoViLa is a general purpose software development platform that has been used in various application areas like simulation of hydraulic systems and automated composition of web services. In the context of this work the goal is to study the applicability of the CoCoViLa software development approach to cyber security simulations.

The CoCoViLa platform has been developed by the Modeling and Simulation group at the Institute of Cybernetics at Tallinn University of Technology as a joint effort since 2004. The author of this dissertation joined the group and CoCoViLa development in 2006. An overview of the full platform, including parts of CoCoViLa and some of its applications developed by other people, is given here as it is necessary to present the contributions of this dissertation.

This chapter is structured as follows. Section 2.1 gives an overview of the CoCoViLa platform and Section 2.2 describes the key principles of the conceptual design of the platform. The specification language is described in Section 2.3. Section 2.4 presents the user interface of CoCoViLa. Section 2.7 contains application examples and finally related work is described in Section 2.8. The contents of this chapter is based on the following papers:

- *CoCoViLa as a Multifunctional Simulation Platform* (Publication I),
- *Enhancing Response Selection in Impact Estimation Approaches* (Publication II),
- *Hybrid Simulation of Large Networks* (Publication III),
- *Rich Components of Extendable Simulation Platform* (Publication IV).

2.1 Introduction

CoCoViLa [31] is a model based software engineering platform that is suitable for performing modeling and simulation tasks. It provides tools for developing reusable software components, constructing packages, specifying simulation problems and running them using pluggable simulation engines. The platform is intended for end users who are domain experts. Therefore it provides visual specification capabilities so that programming skills are not required from end users. CoCoViLa has visual tools, but most importantly, it supports full automatic program construction from specifications that are given visually.

From a user's point of view CoCoViLa consists of three programs with a graphical user interface: *Class Editor* for the development of packages and components, *Scheme Editor* for visual specification of computational problems and for executing generated programs (performing simulations), and *Decision Table Editor* for managing knowledge modules.

CoCoViLa uses a component based software engineering approach. Simulation packages consist of software components called *rich components* [55]. Rich components are descriptions of domain specific concepts that are used in simulation. In a nutshell, these components are composed of the following four parts. First, there is a graphical representation of the concept (visual part) for interacting with the user in a graphical user interface. Having such a representation, schemes of simulation tasks can be built via visual composition of the components. Second, the logical part or *metainterface* containing a high-level specification that enables automatic composition of a simulation program. The specification language is described in Section 2.3. The third part is a program component which defines Java methods that implement computations. This part is called *metaclass*. Finally, a rich component may have a daemon that provides interactive properties allowing to develop sophisticated interfaces to simulation programs, enforce syntax rules of visual schemes, implement agent-like behaviour etc.

It is expected that simulation problems are becoming not only computationally heavier, but also considerably more complex in the sense that a single problem may require orchestrated usage of different simulation engines, external data sources, optimization programs as well as visualization and statistics software. CoCoViLa's visual component based approach supported by structural synthesis of programs provides a solid foundation for developing complex simulations.

2.2 Design Principles

The design and implementation of CoCoViLa has been guided by several design principles that will be summarized here.

First, a main design decision has been to rely on a completely automated program construction approach that starts from a specification and uses a fast synthesis method that outputs a source code ready for compilation and execution. The program synthesis method used in CoCoViLa—structural synthesis of programs (SSP) [70]—has been implemented and used in several earlier software tools [50, 73]. It uses essentially dataflow for composing a program, like, for instance, Simulink [15] does. However, the usage of components in the form of higher-order functions that take synthesized parts of a program as inputs makes a significant difference over conventional dataflow techniques. The program synthesis process uses dataflow recursively for solving so called *subtasks*—generating inputs for the higher-order functions, i.e., for achieving the goals like "synthesize a body for the loop in this particular component". This makes the method universally

applicable—theoretically any algorithm can be synthesized in this way from a suitable specification and a fixed set of preprogrammed components [50]. A program is synthesized piecewise and the pieces are bound together by preprogrammed higher-order functions that realize required control structures. The planner is CoCoViLa's core part responsible for synthesis, it is described in more detail in Section 2.5.

The second design decision is using full capabilities of model based software engineering: developing a complete model for each simulation problem that includes not only a description of a simulated system, but describes also the usage of simulation engines and other tools—visualizers, optimizers etc.— needed for the particular problem. A model based software engineering process consists of two stages: domain engineering that provides assets for developing applications, and application engineering that uses the assets for applications. The first phase requires the work of domain analysts and software developers. In the second phase, domain experts are the main users of existing software assets and the developers of applications. Reusable assets of the CoCoViLa model based software engineering approach are rich components—Java classes extended with specifications for program synthesis and supplied with a visual representation (see Publication IV). A rich component may have another class associated with it that defines a *daemon*—a separate thread that supports user interaction during the problem description and program execution phases. This allows to implement agent-like behaviour in rich components. In the context of CoCoViLa technology and also this dissertation, the term component is often used in place of rich component. A collection of components for a problem domain constitute a *package* that is an implementation of a domain specific language (DSL) for this particular domain.

Third, the Java programming language and platform has been chosen for implementing CoCoViLa. This is justified by several useful properties of the Java ecosystem: availability of solid development tools and a large number of libraries, portability and interoperability, support for distributed computing, open source ideology, dynamic compilation and loading of program code. The CoCoViLa platform has been developed in a way that does not restrict the usage of Java for programming of classes that implement components. Moreover, it was straightforward to integrate the specification language with the Java programming language. The specification language will be introduced in the following section.

2.3 Specification Language

The CoCoViLa specification language is used for describing both components and computational goals. This language is built on top of the Java programming language and merged with Java in the following way: a specification is given as a special comment included in the source code of a Java class. Here is an example of a specification that shows the usage of equations in a specification:

```
class Complex {
```

```
    /*@ specification Complex {
        double re, im, arg, mod;
        mod^2 = re^2 + im^2;
        mod * sin(arg) = im;
    } @*/
}
```

The connection between the specification and the Java code implementing computations is through method names. The following example shows the usage of a Java method (`getMaxVal`) as an implementation of a higher-order function in a specification:

```
class Max {
    /*@ specification Max {
        int arg, val, maxval;
        [arg -> val] -> maxval {getMaxVal};
    } @*/

    public int getMaxVal(Subtask sbt) {
        ...
        return maxval;
    }
}
```

The method `getMaxVal` is given a synthesized method that implements the interface `Subtask` as the argument `sbt`.

Each Java method usable in synthesis must have a specification describing its input and output conditions. These specifications are called *axioms* according to the convention of structural synthesis of programs. The meaning of these axioms can be explained in terms of dataflow. The example above includes the axiom

```
[arg -> val] -> maxval {getMaxVal};
```

This axiom has one input `[arg -> val]` that is a subtask describing a function for computing `val` from given `arg` (this function has to be synthesized by the planner). The axiom has an output `maxval`.

The specification language has a rather simple and conventional syntax. It enables one to specify typed objects and bind them with each other by connecting their attributes by equalities. Numeric variables can be bound also by algebraic equations. Constant values can be assigned to variables of any type, as soon as the value has a textual representation. The textual specification enables one also to specify the usage of methods of the class where the specification is included by writing axioms about their applicability, i.e., by giving their pre- and postconditions. Extensibility of the language is achieved by the introduction of new types. The following is the core of the language.

1. Declaration of a component

   $type\ name$;

   This declaration specifies a component of a model with given type and name.

2. Binding

   $var1.field1 = var2.field2$;

   This statement specifies an equality between fields of components. The fields may be represented by ports.

3. Valuation

   $var1.field = value$;

   This statement defines a functional dependency with no inputs and with one output that receives a constant value.

4. Axiom

   $precondition \rightarrow postcondition\{implementation\}$;

   The precondition of an axiom is a list of component names and subtasks. The postcondition is a component name. The names in precondition show components that are inputs of the computation given by the method name `implementation`. Postcondition shows a component in output of the computation. A subtask has the form $[x_1, \ldots, x_n \rightarrow y_1, \ldots, y_m]$ and defines a function with inputs and outputs given on the left and right side of the arrow. The function defined by a subtask has to be synthesized and given as an input to the function described by the axiom.

5. Equation

   $AExpression = AExpression$;

   Equation defines one or more functional dependencies that are solving functions for variables bound by the equations. Arithmetic expressions are the ones supported by the Java platform, and can be solved only for the variables that have one occurrence in an expression.

6. Tuple

   $alias\ id = (ListOfNames)$;

   $ListOfNames$ can include names with wildcards of the form $*.name$. In this case all fields of components of a specification that have the name $name$ are included in $ListOfNames$. For example, `alias state = (*.state)` describes a state vector consisting of states of components. This feature is very useful in many situations, increasing flexibility and reusability of components. It allows to define new concepts that encapsulate parts of other components dynamically in a natural way.

7. Inheritance

   $A\ super\ B_1, \ldots, B_n$

   The specification language supports object oriented inheritance. The keyword $super$ in the declaration of a concept A causes the declared concept to inherit all variables, bindings and axioms defined in the list of parent concepts $B_1, \ldots, B_n$. The parent concepts are also called *superclasses*.

The visual language describes schemes and, strictly speaking, uses only the first two kinds of statements. However, through the graphical user interface of Scheme Editor (see Subsection 2.4.2) one can add also valuations, aliases, equations and a superclass to a scheme.

A full description of the specification language can be found at the CoCoViLa web site [18].

2.4 User Interface

The graphical user interface of CoCoViLa consists of three applications. First, Class Editor is for creating reusable components representing domain specific concepts and organizing components into packages. Second, Scheme Editor is for composing visual specifications using domain specific concepts from packages. It is also used for defining the goals to be solved and the execution of synthesized programs. Third, Decision Table Editor allows to manage collections of rules used as knowledge modules for expert system features. A brief overview of these applications will be provided below.

2.4.1 Class Editor

In the Class Editor users can define visual aspects of rich components using drawing capabilities or by importing corresponding image files. Figure 2.1 shows the development of a component responsible for plotting charts in the window of the Class Editor. The image of the chart contains two ports (red circles) for providing data to the axis. A smaller pop-up window is for defining properties of a selected port. Another window is for specifying attributes of the given component, e.g., class name, toolbar icon, description and a set of fields of visual interface with types and default values. Functional properties of this component are implemented in a Java class.

In other words, Class Editor supports a language designer in defining visual, logical and interactive aspects of concepts. Moreover, it is a tool for binding visual parts of a component with other parts developed as Java classes. It is also used for developing packages—creating new packages, and importing and exporting components.

The usage of Class Editor is described in more detail in the user manual available from the CoCoViLa web site [17].

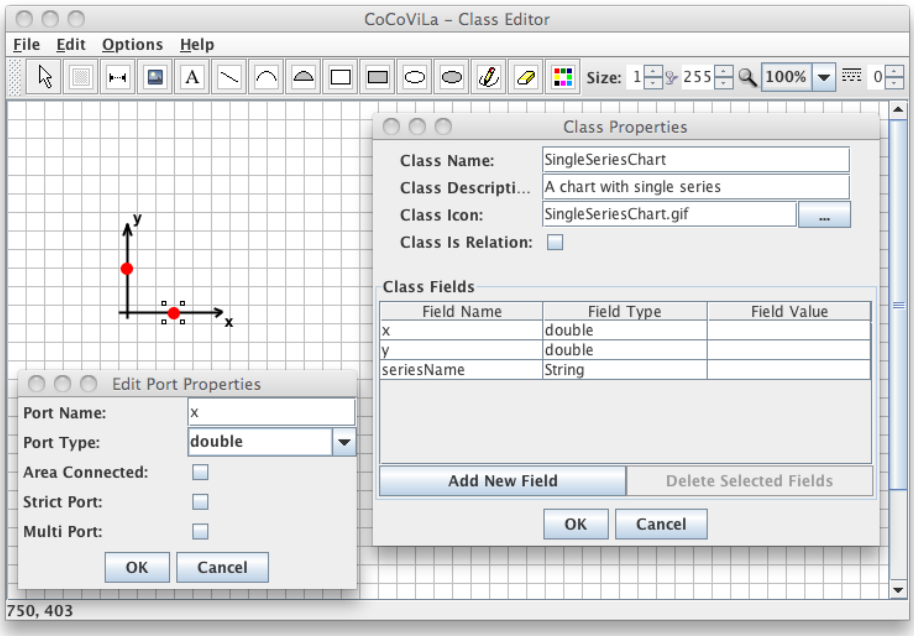


Figure 2.1: Class Editor window (Publication I)

2.4.2 Scheme Editor

The Scheme Editor is a multi-purpose tool. It allows to load a package created in the Class Editor (user interface with toolbars and menus is determined from the package description) and to compose models in the form of visual schemes. Schemes can be exported and used as components in other schemes. This feature allows to model complex systems using hierarchical composition which is important for helping the users to manage complexity.

From a visual description of a problem, an executable program is synthesized automatically. There are also debugging capabilities (algorithm visualizer, viewer for synthesized code, etc.). An executed simulation program can show results both in a separate window or display the feedback directly on a scheme.

Figure 2.2 shows the Scheme Editor in use. In this screenshot a basic package called *Hns* for simulating hybrid queuing networks is loaded. The scheme describes a network of three traffic generators and a bounded buffer. This scheme has been composed by connecting ports of components of the following types: `TrafGen`, `Buf`, `Clock` and `Graph`. The toolbar at the top of the scheme is for adding objects and connections to the scheme. The component *Proc* is a simulation engine that implements a hybrid simulation process. It is specified as a superclass of the scheme indicated by the green background color. A pop-up window and a pop-up menu are also visible in the figure. The pop-up window is for instantiating object attributes,

the pop-up menu is for manipulating the scheme—deleting and arranging objects etc.

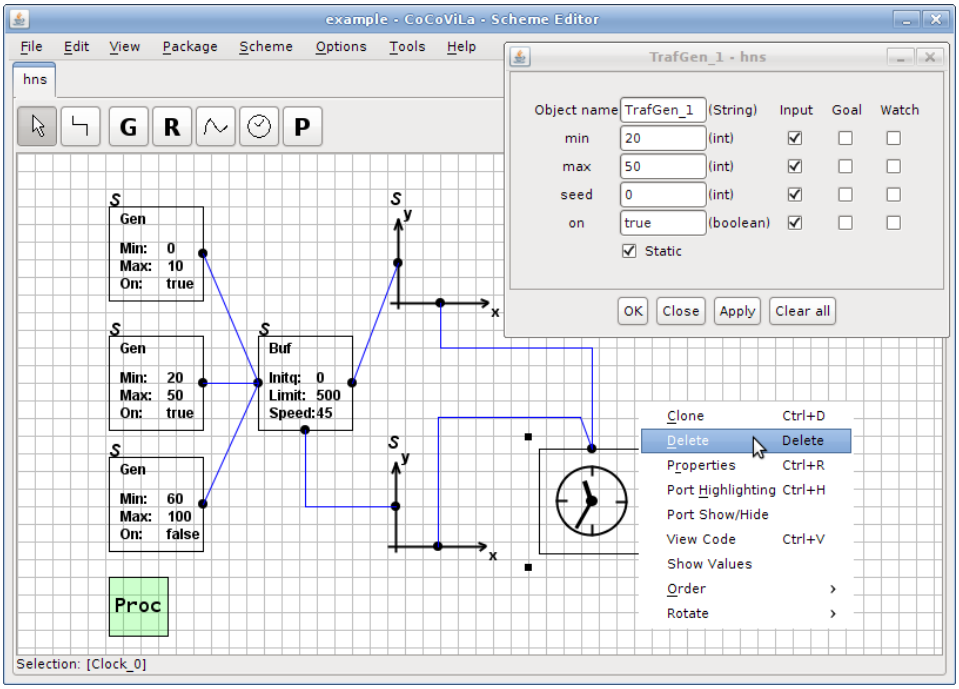


Figure 2.2: Scheme Editor window (Publication III)

Figure 2.3 shows a textual specification obtained from the scheme of the example shown in Figure 2.2. On the left in Figure 2.3, the specification of the class `Hns` derived from the scheme is shown. On the right, the figure shows a fragment of a simulation algorithm synthesized from the specification. This algorithm is compiled into a Java class that in the present case is 144 lines of code and is not shown here. This Java class can be compiled and run in the Scheme Editor. The scheme and the algorithm windows shown in Figure 2.3 are useful for debugging a specification, but they are not needed when solving simulation problems in CoCoViLa, because during normal use the user interacts with a simulation program through the main window of Scheme Editor shown in Figure 2.2.

Additional information about Scheme Editor can be found in the user manual available from the CoCoViLa web site [17].

2.4.3 Decision Table Editor

For many applications it is useful to have the means for handling expert knowledge. To cover this need, CoCoViLa supports decision tables which provide a compact and a convenient way for representing application specific knowledge as sets of rules. The Decision Table Editor is a standalone tool for editing these rule sets.

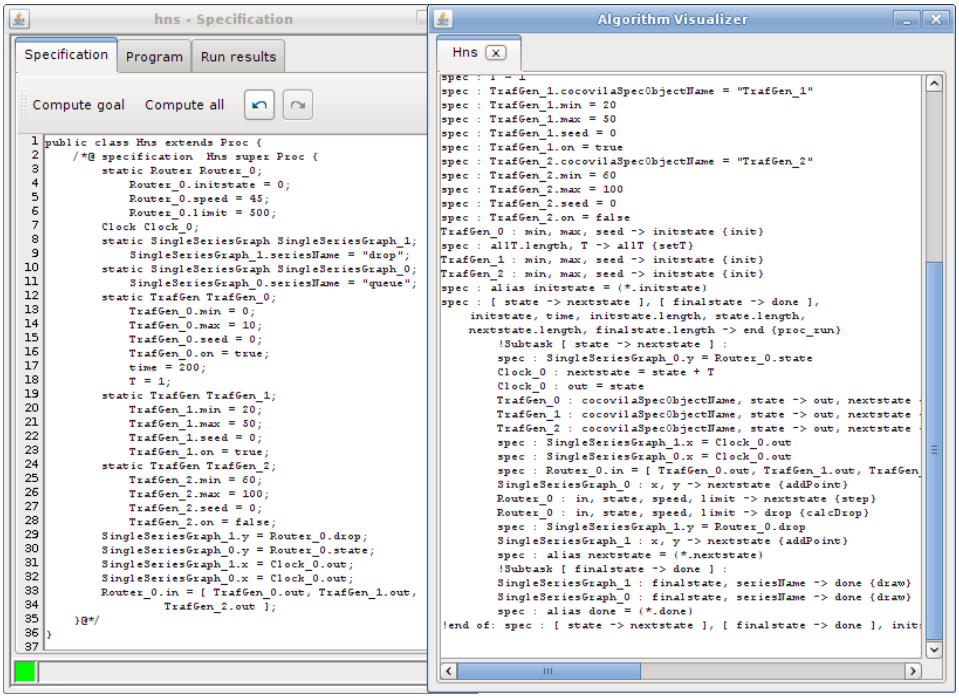


Figure 2.3: Textual specification and synthesized algorithm (Publication III)

Figure 2.4 shows the main window of Decision Table Editor opened in the expert system management user interface.

Several decision tables can be used simultaneously in one scheme. For acquiring data from decision tables, a forward chaining inference engine is used internally. The support for using data from knowledge modules has been integrated into the specification language via a dedicated `@table` keyword.

Further information about the usage of Decision Table Editor can be found from the software user manual [30].

2.5 Planner

The *planner* is a core part of CoCoViLa. Its purpose is to transform declarative specifications of computational problems into executable programs. The planner determines computational paths from initial variables to required goal variables (i.e., tries to solve a given *computational problem* "find values of V from given values of U", where U and V are sets of input and output variables). The planner's task is not only to construct a linear dataflow, but also to solve subtasks (higher-order dataflow) and to perform optimization of an algorithm. Generation of a resulting program's code from an algorithm is then a straightforward process.

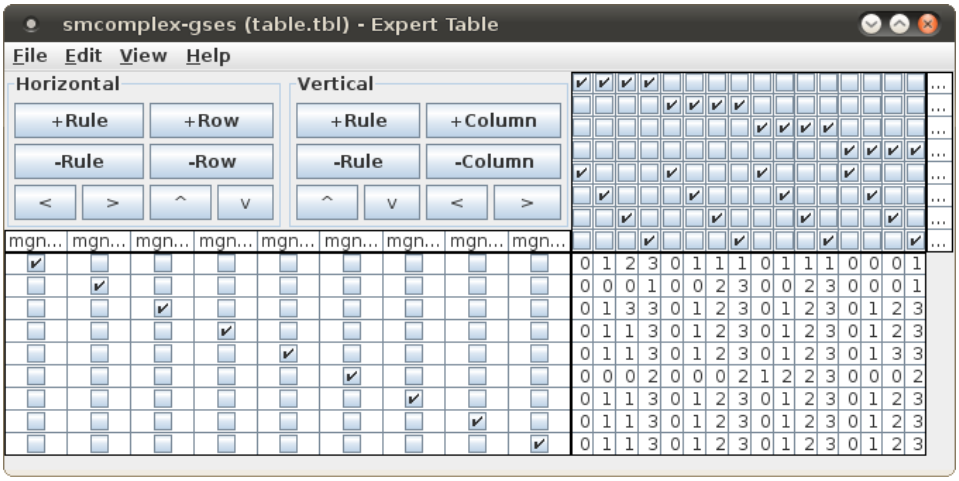


Figure 2.4: Decision table editor (Publication I)

Let us have a small example of dataflow planning where a goal is to compute a next state of a simulation from a current state. The computational problem can be expressed with a following statement:

```
state -> nextstate;                (1)
```

The problem can be solved only if a relation between `state` and `nextstate` is specified. For simplicity, let us assume the state is just a numerical value and the next state is an increment of a current state by a step which is obtained using a method `getStep`:

```
nextstate = state + step;          (2)
-> step {getStep};
out = nextstate;
```

In this case the planner produces the following dataflow if a value of the state is given as input:

```
state = getStep();
nextstate = state + step;
```

Note that the calculation of a variable `out` in the previous listing is not included in the dataflow because it is not required for solving the given problem (1). Another statement including a control variable `printed` can be added into the specification that prints the value of the `nextstate` using the method `print`:

```
nextstate -> printed {print};      (3)
```

The dataflow for (1) will not include (3) because (1) does not contain a control variable `printed` in the set of outputs. To achieve this, another computational problem has to be stated:

```
state -> printed;
```

The example above shows how to construct a linear dataflow, i.e., to compute the value of the next state once. Simulation tasks require to compute states in a loop until some satisfying final state is reached. To specify such a task in CoCoViLa, subtasks have to be used. The following statement specifies that a final state can be computed from a given initial state if there exists a function that calculates the next state from a given state.

```
[state -> nextstate], initstate -> finalstate {proc};
```

To solve a topmost computational problem `initstate -> finalstate`, the subtask `state -> nextstate` must be solved. Having (2), the subtask is solvable and a higher-order dataflow can be constructed by the planner. The synthesized function `state -> nextstate` is passed as an argument to the method `proc` and this method can iteratively call the function to increment the state as long as it is needed.

Following the described technique, simulation engines that require loops and other control structures are implemented using subtasks and the planner takes care of synthesizing bodies of subtasks. In Publication III a hybrid simulation engine is described. This engine supports the execution of continuous processes that also include discrete parts modeled as discrete event simulation.

In addition, the planner can be invoked at runtime for generating new programs to solve tasks on models that might have been changed dynamically during the simulation process.

In general, the synthesis of an algorithm with subtasks has exponential time complexity with respect to the number of subtasks in a specification, as the solvability of one subtask may depend on the solvability on another subtask or it can be the case that one and the same subtask has to be solved repeatedly in one and the same branch. An implemented algorithm is incremental depth-first search with backtracking and additional heuristics, delivering good performance on typical practical problems. More details about the implementation and performance benchmarks are presented in P. Grigorenko's dissertation [29].

2.6 Toolbox

Toolbox is a package in CoCoViLa that contains a number of universal and reusable components useful for building simulation programs. It includes several simulation engines, visualization components and other helper components, such as tables and sliders.

The toolbox includes basic simulation engines as components for driving discrete-event, continuous time and hybrid simulations. Simulation engines are used as superclasses of schemes, but they are not scheme or application specific. Being a superclass, a simulation engine is able to collect parts of a state from the

underlying components automatically using aliases and wildcards. For visualization the toolbox contains several components for plotting diagrams, two- and three-dimensional charts.

In the conventional CoCoViLa domain engineering process users take individual components from the toolbox package and import them into their own simulation packages. Essentially, this means the components are copied from the toolbox and can be later modified as a part of a new package. Due to historical limitations of the platform it has been impossible to develop CoCoViLa applications that use components from more than one package. From the software engineering perspective this results in duplication of development effort and reduces maintainability of packages. In this work an ontology based modular solution is proposed to alleviate this limitation. The solution is described in Chapter 3.

2.7 Applications

The platform has been used for various software engineering and simulation applications. To provide a broader overview of the capabilities of the platform, two applications from different domains are introduced in this chapter. The first one, described in the following section, is about the simulation of specific hydraulic systems. This application is notable for its size and complexity. The second example, presented in Section 2.7.2, is related to cyber security simulations and is intended for the selection of optimal technical countermeasures to security incidents and cyber attacks.

In addition to the applications presented in this chapter, the platform has also been used for other purposes. For example, an application for automated composition of Estonian e-government web services [48, 49] has been implemented. The solution lets the users to think in terms of, and express computational goals using high level domain concepts from in e-government ontologies, such as *Citizen*, *CadastrialNumber* and *NationalIdCode*. The output is a definition of a new complex web service composed by the planner. This package shows the applicability of the platform for general purpose software development, including applications based on services oriented architecture. It demonstrates also the scalability of the approach to models containing several hundreds of objects.

2.7.1 Simulation of Hydraulic Systems

An advanced and a relatively large application of CoCoViLa is a modeling and simulation package for fluid power devices [34]. Fluid power systems assume a lot of drive and control tasks in machinery because of their high power density, flexible system character and required reliability. Computer modeling and simulation is an important phase in the design of such systems.

Multi-pole mathematical models and signal-flow graphs of hydraulic elements are used. This enables methodical, graphical representation of large and compli-

cated chain systems. Simulated systems are decomposed into subsystems and functional elements. Multi-pole models of them may use different programs, depending on the observed process (steady-state condition, frequency characteristics, transient responses). Calculations are performed using multi-level method. In this way large differential equation systems can be decomposed into smaller ones. First, calculations on the level of elements or subsystems are performed, thereafter variables between elements and subsystems are made congruent by iteration methods.

The package contains over hundred multi-pole models of fluid power elements that can be used for composing different schemes of fluid power devices and performing simulations.

An example presented in Publication I considers simulation of a hydraulic load sensing system. In this example, a hierarchically built model of the device includes over 4500 dependencies represented by equations and Java methods. Typically, two kinds of simulations are performed: calculating steady state conditions and dynamic responses. Different simulation engines are used for calculating steady state conditions, 3D simulations and dynamic transient responses.

A special technique is used for calculating variables in loop dependences that can appear when a scheme of hydraulic device is composed from visual components. This technique is called splitting and it takes initial approximate values and uses iterative re-computing of the variables. Re-computing algorithms are constructed by the CoCoViLa planner as a result of solving corresponding subtasks. This avoids solving large equation systems during simulations.

The typical simulating task for calculating transient responses of the load-sensing system contains 37 classes, including 26 functional element classes, and 16 variables that have to be iterated during the computations. The automatically synthesized Java code for solving the simulation task for calculating dynamic transient responses that mainly consists of calls of methods has 4449 lines and includes 4 algorithms for solving different subtasks. Its synthesis takes less than a second on a typical 2 GHz laptop.

The application described above is aimed at giving the end user a convenient tool for experimenting with different model configurations and varying parameters of the models.

2.7.2 Simulation of Automated Cyber Attack Response

CoCoViLa has been applied in the cyber security domain for modeling and simulation of cyber attacks and selection of optimal countermeasures (Publication II). Graph-based Automated Denial-of-Service Attack Response (GrADAR) [38] is an approach where the selection of attack responses is made according to an estimation of an impact of simulated counter-attack measures. CoCoViLa was used to create a GrADAR package for visual modeling and simulation of computer networks on the basis of information such as dependencies between system resources and their

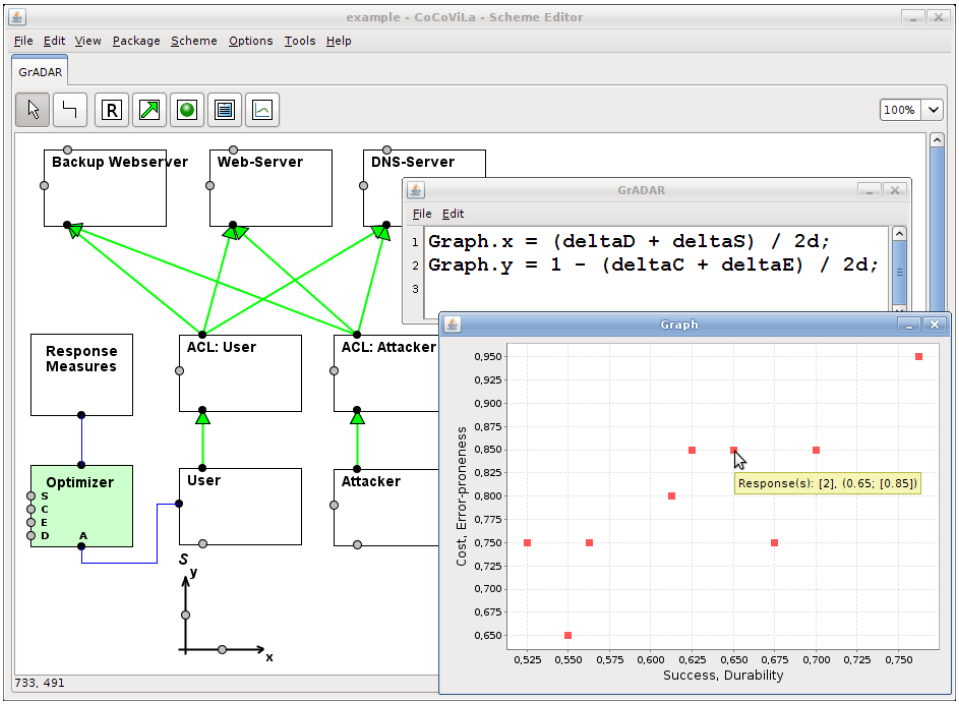


Figure 2.5: Visual specification of a response analysis problem in GrADAR package (Publication II)

availability and workload values. The optimizer component of the package implements algorithms for automatic selection and application of response measures.

Figure 2.5 shows a scheme in CoCoViLa representing a dependency graph of network resources with connections for propagating workload and availability values. The goal is to simulate and analyze the effect of response measures. CoCoViLa allows to enter the parameter values using the graphical user interface or query values from decision tables (see Subsection 2.4.3). Moreover, thanks to the absence of restrictions on the usage of Java language and libraries, it also allows integration with other systems, e.g., monitoring solutions or statistics databases to receive live values into the running program synthesized from the scheme.

One of the potential uses of this application is the support of cyber security operations where quick analysis and decisions support are needed in case of ongoing attacks.

Another cyber security related CoCoViLa application is modeling and simulation of graded security measures. This application is a main part of this work and is discussed in detail in Chapter 4.

2.8 Discussion and Related Work

Modeling and simulation is a broad field and there exist hundreds of software tools and products applicable for modeling and simulation. A paper by Leblanc et al. presents an overview of cyber attack simulation tools [44], another paper by Ouyang reviews tools and techniques for modeling and simulation of critical infrastructure systems [58]. In many cases simulation tools are tailor made for solving a specific problem in a narrow area. Therefore the tools can be very different and may not be even directly comparable. The ones having common properties with CoCoViLa can be roughly divided into three categories: general-purpose modeling and simulation software (e.g., Simulink [15]), model based application development software (e.g., MetaEdit [66]) and large-scale discrete event simulation software (e.g., OPNET Modeler [57], OMNet++ [76]). A universal feature of most of these tools is visual specification capability by drawing schemes from components and connecting them to each other.

General purpose simulation software products (Simulink [15], Scicos [11], Ptolemy II [46], etc.) typically possess built-in hybrid simulation engines and the simulation is flow-based—i.e., all the connecting arcs are directed and all the ports are either inputs or outputs. The models can be composed hierarchically and the components can be easily reused. Simulations are typically carried out on a virtual machine. Also, model translators exist to get code in some programming language (e.g., C) and to improve efficiency running the compiled code. Standard components (blocks) are grouped into packages (palettes). There exists a large variety of blocks and new blocks can be developed to cover missing pieces.

Model based application development tools such as MetaEdit [66] have commonly a rich set of tools for program specification, analysis and verification. Taking the model based approach to simulation development, it is straightforward to compose modeling and simulation applications with these tools. However, it typically needs more effort than using a dedicated simulation tool.

Large-scale discrete event network simulation tools (OPNET Modeler [57], OMNet++ [76], NS3 [60], etc.) are scalable simulators capable of handling large number of nodes and events. Their main concern is simulation performance and high-fidelity modeling of network devices and protocols. These tools are mostly used for in-detail analysis of network behavior, simulation testbeds and embedded systems, simulating all actions down to hardware level.

The CoCoViLa platform itself has been developed bearing in mind programming of simulation problems. However, it has been used also in a more general model based software development, e.g., for composing web services on large service models [49]. In this sense it is similar to visual software development tools like MetaEdit [66].

The CoCoViLa's planner provides flow based analysis similar to Simulink or Scicos, but its capability of constructing higher-order data flows provides additional flexibility compared to other tools, allowing, for example, the development

of model- and simulation-specific simulation engines. As the resulting simulation is always compiled into a single program it allows to achieve good simulation performance.

In comparison to other tools, CoCoViLa's support for handling bindings is very useful as this allows better reuse of rich components. For example, the ports are not defined strictly as inputs or outputs like it is the case with the tools belonging to the first two categories. It is up to the planner to decide which way the data flows in order to match the needs of the given problem—like it is the case also in real life—when defining a pipe it is not known beforehand in which direction the liquid flows in it.

A simulation engine and other supporting tools are usually built into the simulation platform. It is a common practice that a simulation problem itself is described using a model based technology, mainly by specifying a model of the simulated system. CoCoViLa takes this one step further—model based approach is used for constructing and compiling a completely new program for every simulation, using simulation engines, optimizers, visualizers etc. as components. Almost any aspect of a simulation can be customized by replacing a component with another from a library or by implementing a new component. This feature allows to call CoCoViLa multi-functional and makes it generally more broadly applicable than other similar tools. This approach is feasible thanks to efficient automation of program construction from a given model.

A useful feature of the CoCoViLa platform is the ability to generate parts of simulation programs automatically using structural synthesis of programs, even at runtime, if needed. This adds flexibility in developing and reusing simulation models, as, for example, functions (e.g., event handlers) can be synthesized lazily.

When comparing simulation tools there are other significant attributes to consider such as the licensing costs, possibility to get support, availability of source code, extendability, the freedom to modify the tools, the size of user base, availability of documentation, the amount of community developed models and component libraries etc. In case of simulation tools where small implementation details may affect results in unexpected ways the availability of source code and the freedom to inspect and modify it are crucial. This is especially important with security related simulations. Also, for wide adoption it is useful to have a tool with a free license for all applications. Hence, a good choice for simulation software is an open source platform with optional commercial support. The CoCoViLa platform is developed as free and open source software but is currently lacking commercial support offerings and a large enough user base.

2.9 Conclusion

This chapter gave an overview of the CoCoViLa platform and described its features. In this platform a component based approach to software development is

used. The platform provides flexibility by supporting domain specific visual specification languages and allows to implement domain concepts and simulation engines as reusable components. Moreover, a main feature is the support of structural synthesis of programs for translating declarative specifications of simulation problems into executable code. The program synthesis method and the underlying textual specification language provide convenient means for working with domain concepts and defining new hierarchical components in a natural way. Reusable components can be implemented in Java programming language without restrictions. Therefore, existing Java libraries are available, for example, to interface with third party tools, databases, monitoring solutions, etc.

CoCoViLa serves as the infrastructure for supporting the model based software engineering methodology. The extension developed in the scope of this work are discussed in more detail in Chapter 3. This chapter also referred to various CoCoViLa applications in different domains, including the cyber security domain. A cyber security application example for graded security cost optimization will be presented in Chapter 4.

3 OWL Ontologies in DSL Development Process

Domain specific languages (DSLs) provide convenient and efficient means for describing systems using the concepts of a domain. The CoCoViLa software technology is based on user defined visual domain specific specification languages and backed by automated synthesis of programs built into the platform. As discussed in Chapter 1, achieving cyber security goals needs cooperation between different departments of an organization. Situation awareness that requires accurate perception of the environment and comprehension of its semantic structures is also a prerequisite of successful cyber security planning and operations. In order to facilitate communication, interoperability of software tools and reuse of software assets, shared and formalized conceptualizations are needed.

In CoCoViLa, DSLs are implemented as packages consisting of components that represent concepts. CoCoViLa packages are, in essence, shared and formalized conceptualizations of problem domains, similar to formal ontologies. The concepts in CoCoViLa packages carry logical specifications called metainterfaces that express attribute semantics [32] of the models composed of DSL concepts. However, the DSLs developed for CoCoViLa using the conventional approach cannot easily reuse existing semantic assets nor are the resulting packages usable outside of the platform.

To address the issues of communication and interoperability, and to facilitate reusability, a new approach to DSL development that employs semantic web technologies was proposed in this work. The chapter first gives an overview of the model based approach to software engineering. The rest of the chapter is dedicated to describing the proposed DSL development process and the architecture of the prototype implementation. This chapter is based on the following publications:

- *Semi-Automated Generation of DSL Meta Models from Formal Domain Ontologies* (Publication V),
- *Ontology-Based Integration of Software Artefacts for DSL Development* (Publication VI),
- *Semi-Automated Integration of Domain Ontologies to DSL Meta-Models* (Publication VII).

3.1 Using OWL Ontologies in DSL Design

In this work three types of ontologies are introduced into the DSL development process: domain ontologies, system ontologies and DSL meta-model ontologies. This section describes the purpose and gives an overview of these ontology types.

First, the purpose of a *domain ontology* is to provide a specification of conceptualization of domain knowledge. That is, a domain ontology provides the concepts of the domain and defines relationships between concepts. A formal domain ontology could be seen as a static part of a meta-model of a DSL that may also contain references to computational resources. In the DSL design and implementation phase information about domain concepts can be extracted from the ontology. This information can be utilized to generate skeletons (design templates) of visual classes automatically, hence reducing manual work and helping to align DSL meta-models to actual domain concepts.

Second, a *system ontology* describes the modeling language and the concepts of the modeling software system. The purpose of the system ontology is to make the system concepts and relations explicit in order to support interoperability with new and existing semantic resources. For example, resources developed on one modeling platform can be mapped and then automatically transformed for the use in another modeling tool.

Third, a *DSL meta-model ontology* describes a domain specific language by linking one or more domain ontologies to a system ontology and binding domain concepts to system concepts. It can also include references to existing software assets (e.g., software libraries) or general resources (e.g., images, fragments of specification). The concept of DSL meta-model ontology is central to the approach proposed in this work.

In the context of the CoCoViLa technology, ontologies represented in OWL 2 Web Ontology Language [59] are used. OWL is built on the logical foundation provided by Description Logics [5] which is a family of knowledge representation formalisms. The use of OWL brings additional advantages such as interoperability with (semantic) web technologies (existing tools, e.g., Apache Jena), network transparency (generally no distinction between local and remote resources), global naming (Uniform Resource Identifiers), etc. Another benefit of the proposed solution is that it removes in a natural way the current limitation of being able to use components from only a single package (see Section 2.6). Existing packages, such as the toolbox, can imported into the DSL meta-model ontology. When loading the DSL meta-model ontology, all imported ontologies are found and loaded. Furthermore, the use of OWL DL reasoning capabilities help to ensure consistency of the model and helps in debugging.

The general overview of the approach is shown in Figure 3.1. The CoCoViLa system ontology is utilized in the implementation phase of a DSL. Individuals of the system ontology classes together with their property values are used to store knowledge about a particular DSL meta-model. These individuals are defined in the DSL meta-model ontology that imports the CoCoViLa system ontology and one or more domain ontologies. Apache Jena inference engine is applied to the model in order to extend it with derived knowledge. SPARQL [37] queries are used to extract the DSL meta-model from an extended graph representation of the DSL meta-model ontology. Extended graph representation is obtained by running

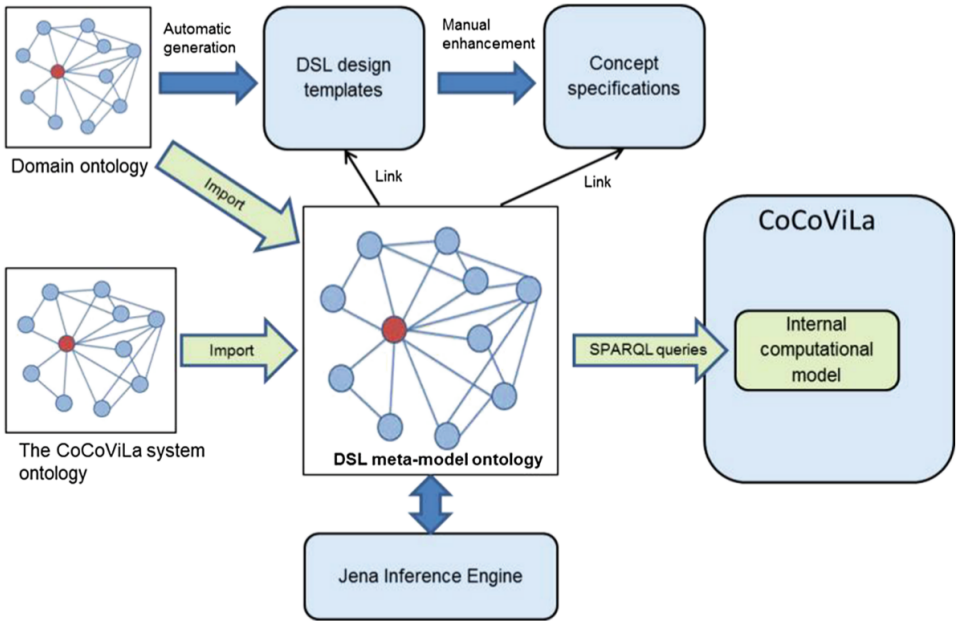


Figure 3.1: Overview of the ontology based DSL development approach (Publication V)

OWL DL inference and applying production rules on the initial graph. Further implementation details are presented in Section 3.2.

For automatic transformation of formal domain ontologies to DSL meta-models corresponding mappings between a set of ontology representation language constructs and a DSL modeling language constructs need to be defined. These mappings have been described in publications V and VII.

3.2 Architecture and Prototypical Implementation

An overview of the CoCoViLa system was given in Chapter 2. Implementing a DSL in the conventional CoCoViLa process required a domain expert to provide the domain knowledge and a programmer able to convert such an informal representation of knowledge into Java classes and annotate these classes with concept specifications. Concept specifications include besides variables also functional dependencies related to concepts. The implementations of functional dependencies can be equations or Java methods implemented in corresponding Java classes. Steps related to the DSL application for solving a particular problem are done automatically by the planner.

A part of this work was to develop a CoCoViLa extension that integrates OWL ontologies described in Section 3.1 into the DSL development process. Figure 3.2 depicts the architecture of the CoCoViLa system extended with ontology based

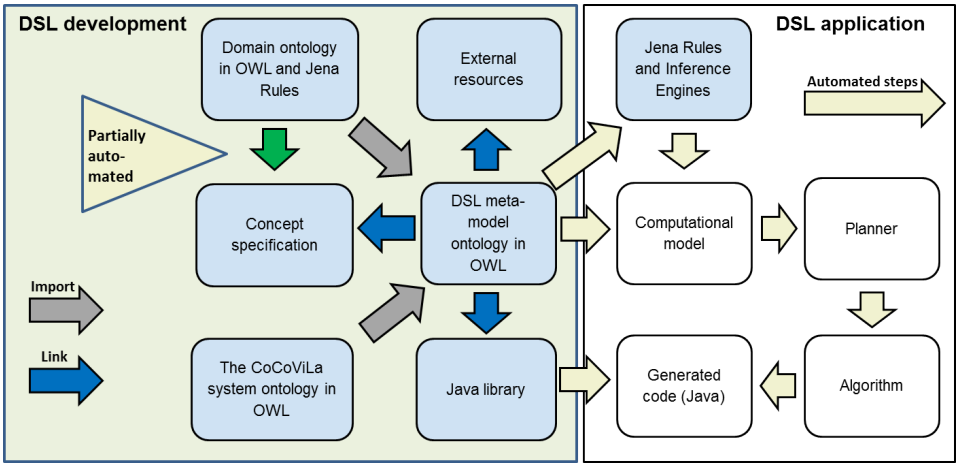


Figure 3.2: The extended CoCoViLa architecture (Publication VI)

DSL development facilities. The extension is mainly related to the improvement of DSL development (domain engineering) while components of the original system are used for a DSL application (application engineering).

The CoCoViLa extension provides facilities for DSL designers to carry out the ontology based DSL development process that enables the usage of existing formal domain ontologies in combination with the system ontology for a DSL construction. When loading a DSL, its meta-model ontology (created by DSL designers) is loaded, extended via Jena inference and rule engine. SPARQL queries are used to dynamically collect and semantically integrate all metadata about artefacts of a DSL meta-model for instantiation of the computational model. Afterwards, the DSL is ready to be used by application developers.

The listing in Section A.1 shows a SPARQL query used in the implementation of the prototype. This query finds individuals representing visual classes. There are five additional queries (packages, fields, all properties, superclasses, and ports) needed for compiling the minimal amount of information required for a usable DSL meta-model.

In this architecture, the internal and external representations of DSLs are decoupled and the mapping from DSL meta-model ontology to internal structures of Scheme Editor is distilled from the CoCoViLa program source code and made explicit as declarative queries.

Application developers build the problem specification using the DSL and translate it into the computational model with the help of the CoCoViLa tool. Applying a set of Jena rules enables to extend the computational model with additional relations between concepts in the model.

Computational model is an internal representation of the computational problem and concept specifications. It is used as an input for the CoCoViLa planner, a theorem prover, which considers the computational model as a logical theorem

with axioms derived from the functional dependencies defined in the specification. Since the prover is based on intuitionistic logic, the solution to the specified computational problem, an algorithm, is extracted from the constructive proof. CoCoViLa generates the Java source code from the algorithm, compiles and executes it at runtime and immediately presents the result of the computation to the user. The generated code can be later (re)used, as it can be saved into the file system.

3.3 Evaluation: IT Security Risk Analysis Domain

In order to evaluate the proposed OWL ontology based approach to DSL development process, a DSL for IT risk analysis was implemented. The DSL realizes a multi-parameter attack tree method [10]. According to this method, attack trees are used to estimate the cost and the success probability of attacks under the assumption of a rational attacker. Elementary game theory is used to decide whether the system under protection is a realistic target for gain-oriented attackers.

This particular DSL was chosen for evaluation because there exists an earlier implementation of a DSL intended for modeling and simulation of IT security risks using attack trees. The earlier version was developed using the conventional CoCoViLa process and it has been used in teaching for practical course work. The new implementation of the DSL is based on OWL ontologies.

An example application for the DSL has been adapted from the original paper [10]. In this example an attack is analysed where the primary threat is a forestalling release of a software product. This threat is related to the situation where a competitor of an IT company steals the developed source code and completes it to its own product.

In the original example the attack tree is static and the outcome is calculated once for a fixed set of parameter values. For a dynamic simulation experiment the attack tree would be recalculated for different parameter values while visualizing the outcome as a graph. Standard components such as `Graph` and `Proc` (simulation engine) from the CoCoViLa toolbox package can be utilized.

3.3.1 The DSL Meta-Model Ontology

From the domain engineering perspective the desired DSL can be expressed by five OWL ontologies. This is illustrated in Figure 3.3 where rectangles reprsesent OWL ontologies and arrows show import relations.

The solution proposed here consists of the following five ontologies:

- CoCoViLaSystem (`ccvl`)—system concepts;
- Toolbox (`tb`)—generic reusable components;
- AttackTree (`at`)—domain ontology for the threat modeling method [10];

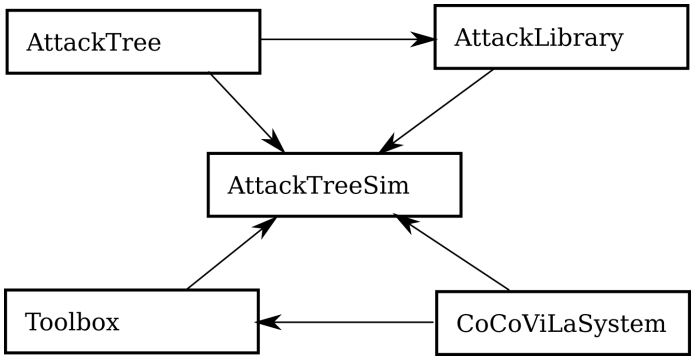


Figure 3.3: Structure of the attack tree package

- AttackLibrary (`alib`)—a library of attack models;
- AttackTreeSim (`asim`)—the DSL meta-model ontology.

The ontologies and their purposes will now be described in more detail in the following paragraphs.

CoCoViLaSystem This is the CoCoViLa system ontology that contains system concepts such as `VisualClass`, `Icon`, `Field`, etc. The ontology version used in this experiment consists of 41 classes in total. It also defines 33 object properties (e.g., `hasPort`, `isImplementedBy`) and 19 data properties (e.g., `hasIcon`, `hasVisualClassName`). The properties have their domains and ranges specified where applicable. This allows to use a reasoner for classifying individuals defined in other ontologies without specifying their classes explicitly. For example, the range of object property `ccvl:hasPort` is asserted to be `ccvl:Port`. That is, from the axioms:

```
ObjectPropertyDomain(ccvl:hasPort ccvl:VisualClass)
ObjectPropertyAssertion(ccvl:hasPort a:vc1 a:port1)
```

it follows that: `ClassAssertion(ccvl:VisualClass a:vc1)`. This derived fact is used in SPARQL queries for finding visual classes.

Toolbox This ontology imports the CoCoViLa system ontology and defines a set of reusable components in CoCoViLa terms. The components are defined as named individuals in the ontology. Datatype and object properties are assigned to the individuals. The individuals are properly classified automatically by OWL DL inference based on property domains and ranges defined in the system ontology. For this experiment a minimal toolbox was developed that consists of three components: `Clock`, `Process`, and `Graph`. The full definition of the toolbox ontology is given in Section A.2.

AttackTree The ontology defines the following concepts from the risk analysis method's domain (see Figure 3.4): `Threat`, `AtomicThreat`, `AndNode`, `OrNode`, and `PrimaryThreat`.

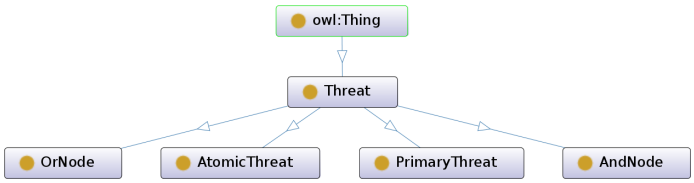


Figure 3.4: Attack tree domain ontology

The concept `Threat` is the domain of the following data properties: `gains`, `hasAveragePenaltyFailure`, `hasAveragePenaltySuccess`, `hasCost`, `hasSuccessProbability`, `threatDescription`. These data properties are automatically mapped to fields of visual classes in CoCoViLa (see Publication V).

AttackLibrary The attack library contains a set of defined attacks with known parameter values. It has imported and uses concepts from the `AttackTree` domain ontology and is independent of CoCoViLa system ontology. In this simplified example the ontology contains just one individual representing a threat: `robberObtainsTheCode`. This individual has values defined for its data properties that will be used as defaults in applications. Figure 3.5 shows the definition of attack library ontology used in this example.

The values specified in the ontology can be seen in the object properties window in Figure 3.7. Default values can be edited on the scheme via the object properties window.

AttackTreeSim This is the DSL meta-model ontology that imports `AttackTree`, `AttackLibrary`, `Toolbox` and `CoCoViLaSystem` ontologies and defines individuals representing visual classes along with the parameter values. Figure A.1 and Figure A.2 give an impression of the size of the ontology and the amount of relations with and without inferred axioms. The DSL meta-model ontology can be loaded in CoCoViLa Scheme Editor as a package and it results in a visual language represented by the icons in the horizontal toolbar (palette) in Figure 3.6. The use of this DSL is illustrated in the following section.

3.3.2 Example Application

Figure 3.6 presents a screenshot of the CoCoViLa Scheme Editor where the DSL meta-model ontology `AttackTreeSim` is loaded. The concepts of the visual language are visible in the horizontal palette (toolbar) as icons. The palette also contains tools for selecting objects and connecting ports. The icons

```
Prefix(:=<http://www.example.org/ontologies/2016/attacklib#>)
Prefix(at:=<http://www.example.org/ontologies/2016/attacktree#>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)

Ontology(<http://www.example.org/ontologies/2016/attacklib>
Import(<http://www.example.org/ontologies/2016/attacktree>)
Annotation(rdfs:comment "Library of attacks")

Declaration(NamedIndividual(:robberObtainsTheCode))

ClassAssertion(at:AtomicThreat :robberObtainsTheCode)
DataPropertyAssertion(
    at:hasAveragePenaltyFailure :robberObtainsTheCode "100000.0"^^xsd:
        double)
DataPropertyAssertion(
    at:hasAveragePenaltySuccess :robberObtainsTheCode "100000.0"^^xsd:
        double)
DataPropertyAssertion(at:hasCost :robberObtainsTheCode "1000.0"^^xsd:
    double)
DataPropertyAssertion(
    at:hasSuccessProbability :robberObtainsTheCode "0.5"^^xsd:double)
DataPropertyAssertion(
    at:threatDescription :robberObtainsTheCode "Robber obtains the code")
)
```

Figure 3.5: Minimal attack library ontology

from left to right and their origin (indicated by prefix, if not built-in): selection tool (built-in), connection tool (built-in), `tb:Process`, `tb:Graph`, `tb:Clock`, `alib:RobberObtainsTheCode`, `at:PrimaryThreat`, `at:OrNode`, `at:AtomicThreat`, and finally `at:AndNode`.

An example application is developed using this DSL. The goal is to specify the attack tree example given in the original article [10] and to calculate the outcome of this tree over a range of values of the global parameter `gains`. The visual specification of the simulation task is shown in Figure 3.6. In addition to the attack tree, this scheme contains a `Process` object for running the simulation process, a `Clock` object for changing the value of the `gains` parameter during the simulation, and a `Graph` for visualizing the outcome. The result of a simulation as a graph can be seen in Figure 3.8.

3.3.3 Advantages and Limitations

The proposed approach in general and the separation of the DSL into five ontologies has several significant benefits, including the following. First, domain concepts are defined explicitly in a separate ontology that is independent of any system specific definitions. The knowledge extraction process is separated into SPARQL queries. Second, components from the toolbox can be reused by just declaring the URI of the ontology as an import and all required resources can be loaded from

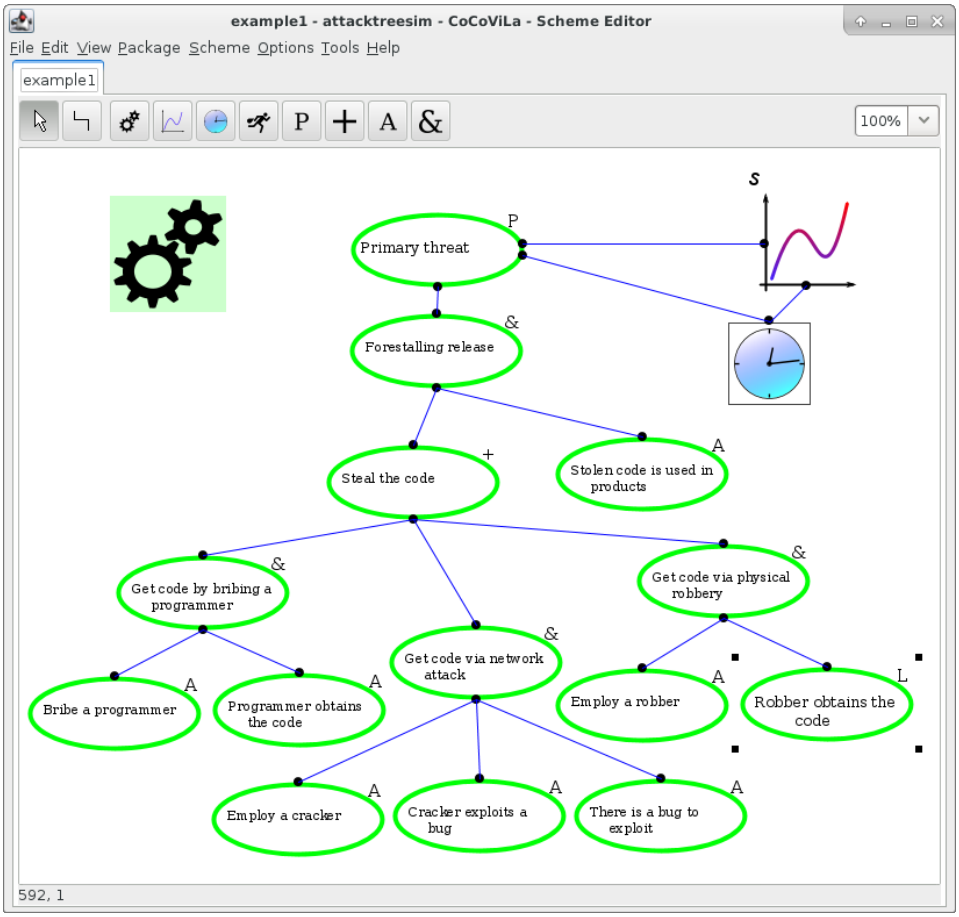


Figure 3.6: An example attack tree

the network. Third, there is no need to specify all axioms manually as some axioms can be derived. For example, if and individual `someThreat` has the property `hasComponentSpecification` defined and the domain of this property is `MetaClass`, then OWL DL inference engine derives the fact that the individual `someThreat` is an instance of `MetaClass`. Fourth, a general purpose ontology validator and a reasoner can be used for validating the DSL meta-model ontology for finding inconsistencies, type mismatches etc. This helps to find and avoid mistakes when developing a DSL. Finally, the use of semantic resources lays a foundation for developing powerful tools that can automate refactoring and development steps.

Currently, there are also limitations. The prototype implementation is functional, but not ready for production use as it does not attempt to cover all use cases that would be needed in practical use. Implementing a production-ready system would need significant effort. CoCoViLa tooling lacks a good user interface for the DSL meta-model development process. For this reason the author used the

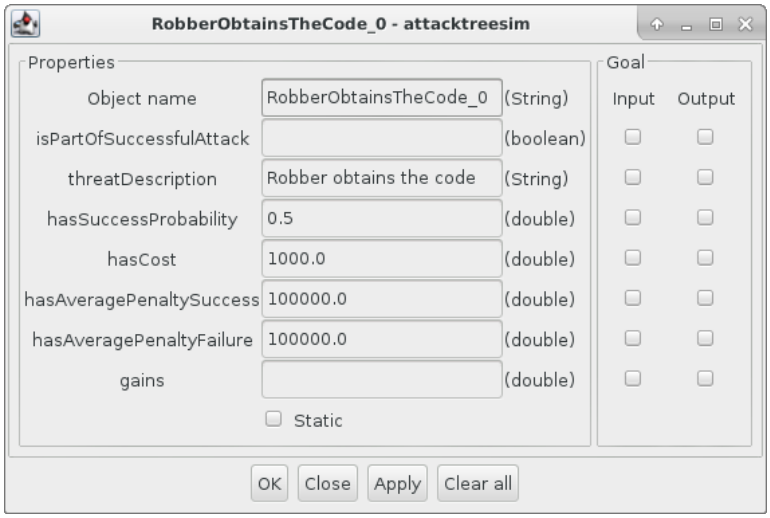


Figure 3.7: Properties of an attack imported from library

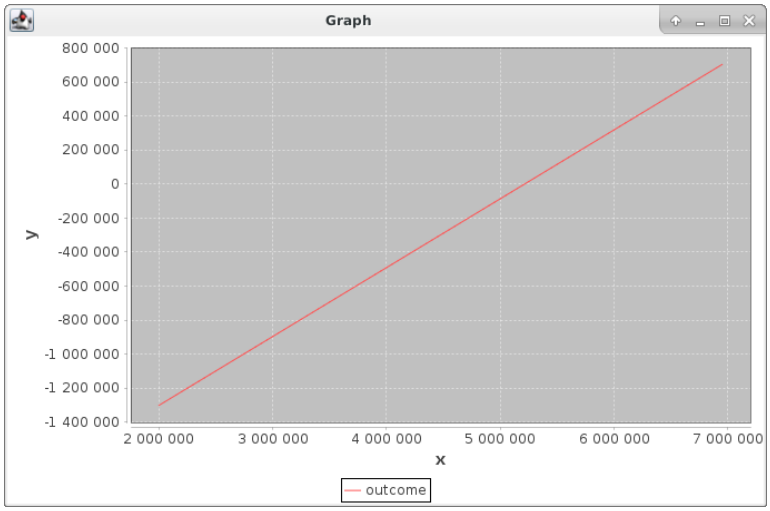


Figure 3.8: A graph of simulation results

Protégé ontology editor [22] instead for the development of this example. Protégé has a comprehensive support for OWL ontology editing. While usable, this process involving external tools is not friendly enough for end-users and would need better software tool support to provide a more integrated user experience.

3.4 Conclusion

This chapter presented an approach that introduces OWL domain ontologies into the DSL development process. A central concept of this approach—DSL meta-

model ontology—was introduced. This approach allows to automatically generate design templates of a DSL meta-model that are consistent with a given domain ontology represented in OWL DL.

The proposed approach creates several benefits, including formal consistency checking of domain knowledge and DSL meta-model ontology, automated generation of design templates of a DSL meta-model that facilitates tracking the evolution of the domain in a DSL. Moreover, it is possible to apply knowledge derived by ontology inference, the use of semantic web technologies improves interoperability and removes some existing modularity and reusability limitations, the use of formal domain ontologies in the early development phase facilitates communication and consistency.

A prototype of this approach was implemented based on the CoCoViLa platform. To evaluate the approach, a DSL meta-model prototype for modeling attack trees was developed. This work demonstrates the applicability of formal ontologies for semantic integration of software artefacts for building DSL meta-models.

4 Graded Security Expert System

This chapter presents the concept of rational security design and describes a methodology for optimal long-term planning of security investments. A supporting tool implemented using the CoCoViLa technology is also presented along with an approach to teaching the methodology to security managers. This chapter summarizes a series of conference papers:

- *Graded Security Expert System* (Publication VIII),
- *Pareto-Optimal Situation Analysis* (Publication IX),
- *Managing Evolving Security Situations* (Publication X),
- *Enterprise Security Analysis and Training Experience* (Publication XI).

The graded security model was originally conceived on the conceptual level by Dr Kivimaa. The author of this dissertation formalized the model, designed and implemented the graded security expert system. In further developments on the model described in this chapter, we worked together both on the conceptual and formal model. The implementation including the optimization algorithms was developed by the author of the dissertation.

4.1 Introduction

Security is expensive. How much should an organisation spend resources on securing an IT system? It is possible to invest large amounts into securing a system by, e.g., implementing increasingly advanced security technologies and training personnel. At some point, though, diminishing returns set in and allocating more resources becomes wasteful. In general, it is unreasonable to spend more resources on protecting a system than the total worth of the system or the potential damage arising from a compromise of the system.

In most organisations the security budget is limited and aiming for absolute security is unrealistic. However, there might be regulations in effect that prescribe a certain baseline security level for a given IT system. Security standards such as NISPOM [54], ISKE [19], BSI [27] and Common Criteria [14] have been developed to ensure a baseline security of complex IT systems. The standards prescribe security requirements as a set of graded security measures depending of the nature or security class of the system to be secured, without including economic

parameters—the costs of implementing the measures. Implementing security measures to cover a full set of security requirements imposed by standards or regulations may be problematic when the budget is limited. In that case priorities have to be set and more important measures have to be handled first. Setting optimal priorities manually for a large set of requirements is complicated and may require a time consuming cost-benefit analysis.

An important aspect of planning and implementing security measures is the time it takes. Comprehensive and detailed security cost-benefit analysis may require months [28, 63]. Often, however, a coarse-grained but quick and good enough solution is preferred to an almost perfect plan after months of effort.

The methodology described in this work provides a rational way of planning security measures and calculating costs. This approach allows to model evolving security situations and to find optimal security solutions based on previous investments and available budgets. Also, the solution includes priorities of security measures, that is, new investments are planned in an order that permits to achieve the best possible security confidence level given the amount of available resources.

Modeling and simulation provides the means to analyse a key question in enterprise security planning of what would be a *reasonable* amount of resources to invest into security and how to justify the decisions. Visualizations help to communicate the results. The method along with the software tool is primarily intended for two purposes. First, it can be used in the security management as a means for decision support, and second, it can be valuable in education for teaching security specialists.

4.2 Rational Security Design

Security planning for an enterprise is aimed at finding a distribution of resources for security measures to achieve the best security solution under given conditions. This is called *rational security design*. Inputs for the rational security design are: *total amount of available money (maximal allowed costs), characterization of the secured system, security requirements.*

Finding a security solution assumes using a security metric that offers a possibility of comparing different solutions, and the possibility to express the quality of the solution by a numeric value. An *integrated security confidence* is used for this purpose that takes into account effects of applied security measures (see the following section). Although security planning is performed as an optimization process, the input data is approximate and mainly based on empirical observations and expert knowledge. Therefore the result is called *rational* and is not necessarily an optimal solution.

Security planning may require solving several partial problems: evaluating costs required for implementing a selected solution, estimating the relative importance

of any taken security solution, detecting similar security measures and collecting them in security measures groups (SMGs), etc.

4.3 Graded Security Model

The graded security model is intended to help to determine reasonable or optimal sets of security measures according to the given security requirements. The model describes how *security measures* are related to *security goals*, what are the *costs* to apply security measures, and what is the *confidence* for guaranteeing the respective *security level*. The essential concepts, variables and function used in the graded security model will be introduced in this section.

The overall security of a system is described by a *security class*. It shows how the *security goals* (confidentiality, integrity, availability, ...) are satisfied. It is determined by assigning *security levels* to security goals, and is denoted by a respective tuple of pairs, e.g., `C2-I1-A1-M2` for the system that has the second level of confidentiality *C*, the first level of integrity *I* etc.

To achieve the security goals, proper security measures have to be taken. There may be a large number (hundreds) of measures. It is reasonable to group them into security measures groups $g_1, g_2, \ldots, g_n$. The grouping should be done in such a way that measures of one and the same group will always be used for achieving one and the same level of security. A function f produces a set of required security measures $f(l, g)$ for a given security measures group g and a security level l of the group. A security class determines the required security level for each group of security measures.

Let s denote a respective function that produces a security level $s(K, g)$ for a group g when the security class is K. An *abstract security profile* is an assignment of security levels $(0, 1, 2, \text{ or } 3)$ to each group of security measures. This can be expressed by the tuple $p = (s(K, g_1), s(K, g_2), \ldots, s(K, g_n))$, where p denotes the abstract security profile and the elements of the tuple p are indexed and appear in the tuple in the same order as the groups of security measures $g_1, g_2, \ldots, g_n$ have been indexed. Knowing the cost function $h(l, g)$ that gives the costs r required for implementing security measures of a group g for a level l, one can calculate the costs of implementing a given abstract security profile:

$$costs(p) = \sum_{i=1}^{n} h(l_i, g_i)\,,$$

where $p = (l_1, l_2, \ldots, l_n)$.

The goal is to keep the value $costs(p)$ as low as possible, guaranteeing a required security. It is assumed that by applying security measures, one achieves security goals with some confidence. The security confidence c of a group g that satisfies the security level l is given by a function $e(l, g)$ and it is a numeric value between 0 and 100 for each group of security measures.

4.3.1 Security Metrics

The graded security model uses coarse-grained metrics differentiating usually three or four security levels for each security goal. To compare security situations in general, one needs a more precise metric that expresses the quality of a security situation by one numeric value. It is reasonable to take into account influences of all security measures on the overall security of the system.

The simplest choice would be to calculate the mean security confidence of all groups. However, the influence of groups on the overall security is different. Therefore, the best solution would be to use partial derivatives of the security measure depending on the security confidences of the groups. These derivatives could be used as coefficients of the security confidences when calculating their mean value. Unfortunately, these derivatives are hard to determine. Instead of the derivatives, one can use empirically found weights of the security confidences.

The overall security of a system is described by means of an integrated security metric S that is a *weighted mean security confidence*, also called *integral security confidence*:

$$S = \sum_{i=1}^{n} a_i c_i \,,$$

where c_i is security confidence of the i-th security measures group, a_i is the weight of the i-th group, and

$$\sum_{i=1}^{n} a_i = 1 \,.$$

Using a linear combination of security confidences of measures groups is reasonable as long as a security situation does not change too rapidly.

4.3.2 Evolving Security Situations

To support long term planning, the model has to include former investments that influence the outcome of the planned investments as well as recurring costs of maintaining already achieved security levels. The following extension to the graded security model allows to take already achieved security level into account in the optimization process. The extended model also accommodates recurring costs that may be implied by some security measures such as licensing costs of antivirus software.

Let us fix a security measures group and consider only one group of security measures here. Then, a simplified form of the functions h and e can be used for calculating costs r and security confidence c—without showing explicitly the security measures group:

$$\begin{aligned} r &= h(l) \,, \\ c &= e(l) \,. \end{aligned}$$

A function h^{-1} is used for calculating security level l for invested costs, which is an inverse function of h:

$$l = h^{-1}(r)\,.$$

Data is needed for already existing security:

$$\begin{aligned} l' &- \text{existing level of security,} \\ c' &- \text{existing security confidence.} \end{aligned}$$

To continue the analysis of security investments, a function H is defined that calculates the additional investments r needed depending on the existing security level l' and the required security level l:

$$r = H(l, l')\,.$$

Instead of the function H another function h^* could be used that calculates the required resources for increasing security level by Δl, where $\Delta l = l - l'$:

$$r = h^*(\Delta l)\,.$$

In the case when no investments in the security have been done before, i.e., when $l' = 0$, the function h^* coincides with the already known function h. However, in the case of $\Delta l = 0$ and $l' > 0$ the degradation of security has to be considered as well—the security level will decrease with time unless additional resources are invested. This shows that the usage of h^* instead of H would be quite a rough approximation.

This analysis is valid for all security measures groups. In the general model an argument g (group number) has to be introduces in each function considered here. This gives the following functions:

$$\begin{aligned} r &= H(l, l', g)\,, \\ r &= h^*(\Delta l, g)\,. \end{aligned}$$

These functions should be obtained from expert knowledge.

Another approach would be to use security confidence c instead of security level. These variables are bound by the function e in the graded security model:

$$c = e(l)\,.$$

The relation between costs and security confidence is expressed by the formulas:

$$\begin{aligned} r &= h(e^{-1}(c))\,, \text{ and} \\ c &= e(h^{-1}(r))\,. \end{aligned}$$

Knowing the already achieved security confidence, additional investments for achieving the new security confidence (or keeping the required confidence level)

can be calculated. This requires the knowledge of a new function E that gives the costs r for achieving required security confidence c by upgrading the given security confidence c':

$$r = E(c, c') .$$

As discussed above, it can sometimes be assumed that the costs depend only on the difference Δc of security confidences:

$$\Delta c = c - c' ,$$

and use the function e^* that calculates the costs:

$$r = e^*(\Delta c) .$$

Again, in the general model an argument g (group number) has to be introduced in each function considered here. This gives the following functions for calculating costs in the general case:

$$\begin{aligned} r &= E(c, c', g) , \\ r &= e^*(\Delta c, g) . \end{aligned}$$

For taking into account the legacy security measures in calculating resources required for achieving a given security confidence, one of the functions H, h^*, E or e^* is needed. It is preferable to use H or E, because these describe the security situation more precisely. In practice, these functions are represented in a tabular form as expert knowledge. Solving the inverse problem—calculating achievable security confidence for given resources—can be done using one of the inverse functions H^{-1} or E^{-1} representable by the same tables as H and E:

$$\begin{aligned} l &= H^{-1}(r, l', g) , \\ c &= E^{-1}(r, c', g) . \end{aligned}$$

The functions H, h^*, E, e^*, H^{-1} and E^{-1} are called legacy functions in this model. The legacy values of l and r are bound by the functions h and h^{-1} as follows:

$$\begin{aligned} r' &= h(l') , \text{ and} \\ l' &= h^{-1}(r') . \end{aligned}$$

Therefore, legacy resources r' can be used instead of l' as inputs of the calculations.

4.4 Expert System

An expert system with visual specification language for security situation description has been built on the basis of the CoCoViLa platform described in Chapter 2. The conceptual architecture of the expert system is show in Figure 4.1. The system

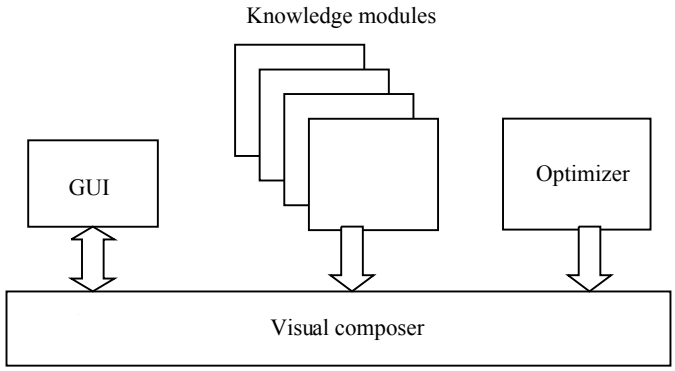


Figure 4.1: Conceptual architecture of the graded security expert system (Publication VIII)

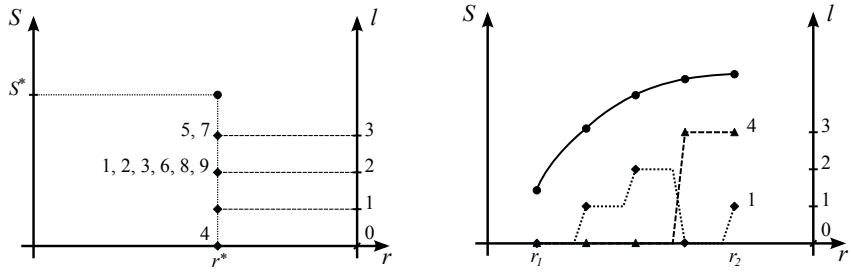
includes knowledge modules (rule sets) in the form of decision tables for handling expert knowledge of costs and gains, as well as for selecting security measures for each security measures group depending on the required security level. Other components are an optimization program for calculation Pareto optimality curve parameterized by available resources, and a visual user interface for graphical specification of the secured system, visual control of the solution process through a GUI, and visualization of the results. These components are connected through a visual composer that solves the specified problems on the request of the user.

The quality and usefulness of the results that can be produced using the method rely on accurate input data—expert knowledge. Gathering and producing high quality formalised expert knowledge is a complex problem on its own and is outside of the scope of this work. The developed solution provides technical means to represent, store and use existing expert knowledge.

4.5 Optimization

The expert system allows to solve several security related optimization problems. First of all, it enables one to find an optimal security solution for given resources, and to determine the reachable security class. This problem concerns again only one value of resources, and can be illustrated by the same picture as the conventional graded security problem (Figure 4.2a).

To get a broader view of possible solutions, one should look at the optimal security for many different values of usable resources. This service is provided by the expert system by plotting a Pareto optimality tradeoff curve that binds resources and the achievable security S. Figure 4.2b shows this curve for an interval of resources from r_1 to r_2. The last value of resources r_2 can be easily calculated as the resources required for getting the security class `C4-I4-A4-M4`. The curve shows



(a) Conventional graded security solution

(b) Pareto-optimal solutions

Figure 4.2: Conventional graded security solution and Pareto optimality tradeoff curve (Publication X)

also the respective security levels for selected security measures groups—in the present case, for the groups number 1 and 4. The exhaustive search of optimal solutions for q possible values of resources, n security measures groups and k security levels requires testing (calculating weighted mean confidence) of qk^n points.

Building optimal solutions gradually, for $1, 2, \ldots, n$ security measures groups allows to use discrete dynamic programming [71] and to reduce considerably the search time. Indeed, the fitness function S defined on intervals from i to j as

$$S(i, j) = \sum_{s=i}^{j} a_s l_s$$

is additive on the intervals, because from the definition of the function S we have

$$S(1, n) = S(1, s) + S(s, n), 1 < s < n \ .$$

This means that an optimal resource assignment to security measures groups can be built gradually, as a path in the space with coordinates x_1, x_2, where x_1 equals to the number of security measures groups that have got resource (i.e., $x_1 = s$) and x_2 equals to the amount of used units of resources. This algorithm requires testing of q^2nk points (q is number of possible values of resources, n is number of security measures groups and k is number of security levels).

4.6 Example

A simplified example will be shown next to illustrate the use of the system. Currently, there are four main security goals in the model: confidentiality (C), integrity (I), availability (A) and satisfying mission criticality (M). For each goal, a certain level (0-3) is attached to specify the security requirements defined by a particular security class. As a simplified example, to achieve the security goals, the following groups of security measures might be selected for consideration: user training,

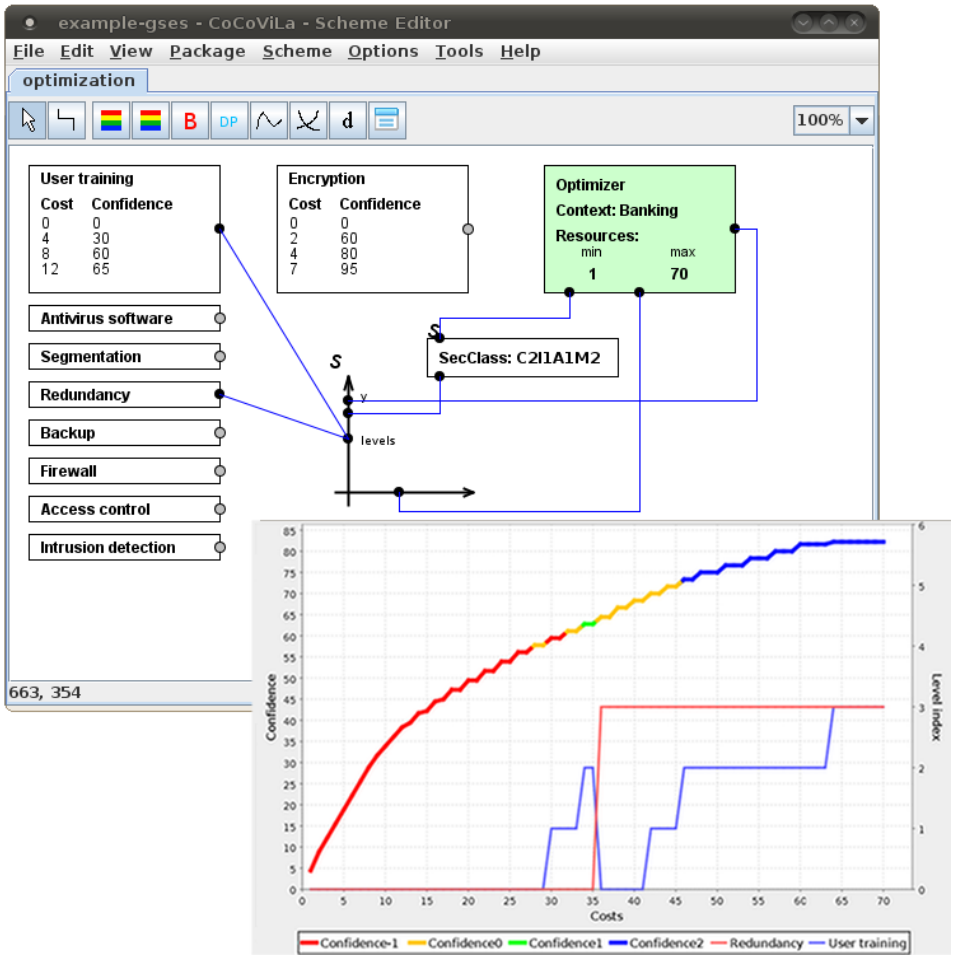


Figure 4.3: Visual specification and the result of simulation (Publication VIII)

antivirus software, segmentation, redundancy, backup, firewall, access control, intrusion detection, and encryption. The relative importance of a measures group can be specified by giving a weight value. In realistic scenarios studied so far the number of measures groups has been between 30 and 40.

Figure 4.3 shows a screenshot of the graded security expert system which consists of a visual specification language, implemented optimization algorithms and knowledge modules in the form of decision tables. In this package, security situations are described using the visual language. Suitable optimizers are applied to simulate all possible outcomes to find the Pareto-optimal set. During the simulation the optimizer queries decision tables for specific values. In the lower right corner of the figure the result of simulation, in the form of a Pareto curve and resource distribution curves, is shown.

As optimization procedures are contained in regular rich components, the algorithm used for a computation is easy to change by just replacing a component on the scheme. It can be useful to have a package contain several optimizers. Then the one making the most appropriate trade-offs for a particular experiment can be chosen by the user. In case of this demo package two different optimizers have been implemented: one employing brute-force and the other discrete dynamic programming algorithms.

Larger examples can be found from Dr Kivimaa's dissertation [41].

4.7 Training Process

The expert system is intended for building security models and solving security design problems: calculating the best distribution of given resources, checking reachability of security goals, planning the evolving security for several years, etc. This tool can be used for training security managers by offering a hands-on experience with solving these problems in a training process.

Trainees are expected to solve a number of security design problems in the conditions that are close to a realistic security planning situation. A suggested order of training steps for these problems is shown in Figure 4.4. The first two are introductory steps. The third step—analysis of security measures groups is an activity for experts. So are the following two steps of adjustment of parameters. Default parameters given in the expert system can be used in training of general managers. Calculating the best distribution of resources is the main training activity. It is possible to concentrate on this activity immediately after introductory steps. Calculating the evolving security continues the previous step and considers security as a process continuing for several years. Checking the reachability of security goals is an independent problem, but it can be solved by the same means as calculating the best distribution of resources.

If a scheme from the scheme library is used, then the first five training steps can be omitted, and one can immediately start solving the last three problems that are training steps for managers as well as for security experts. The model based security analysis is suitable for security training of people with different level of experience. It relies on analytic capabilities of a person, and it introduces the basics of security in the form of a security model. The hands-on part of the training is performed using an expert system. The aim of the presented security model and of set of exercises is to help the trainee to build his or her mental model of cyber security by combining 1) analytical approach where basic concepts and an explicit security mode are introduced, and 2) hands-on experience by solving security design problems with the help of an expert system.

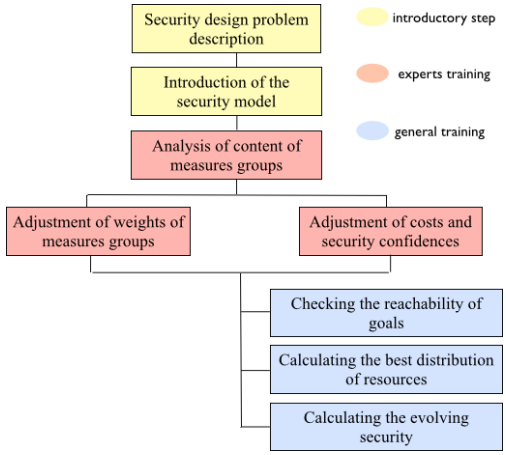


Figure 4.4: Ordering of training steps (Publication XI)

4.8 Related Work

Graded security policy is the basis for security protection programs of nuclear security and anti-terrorist security of US Department of Energy [47, 75], as well as for several European information assurance standards. In particular, German BSI-Standard 100-2 [27] and Estonian ISKE standard [19] are both based on the graded security policy. This policy has been adopted in the graded security expert system and the respective security model has been used.

There is a large number of methodologies and frameworks available for evaluating IT security risks. A common problem is that detailed security risk assessment is complicated and time-consuming leading to organizations not meeting their demands [63]. A paper by Patterson et al. [61] presents an approach to cyber security cost optimization where individual attacks are decomposed into four elements: prevention, detection, mitigation, and consequences. The model is probabilistic and allows to optimize defence measures to meet cost limits. Matlab Optimization Toolbox is used for implementing the optimization procedure. Compared to the graded security expert system, the analysis is more detailed and depends on more accurate data.

The graded security expert system presented here has been developed further and extended with a new optimizer component utilizing an evolutionary algorithm [40]. This illustrates the flexibility and extendability of solutions based on the technology proposed in this work.

Simulation has been found to be effective support in teaching. This is also true in cyber security [26]. Moreover, the need to focus on experimental learning techniques, including virtual labs and simulation, for building conceptual, tactical and

practical skills among cyber security professionals has been emphasized [1]. This work aims to also contribute to this field.

A popular way to teach security concepts is to give some hands-on experience by targeted video games like CyberSiege [65] from the Naval Postgraduate School, or using games from the DISA online training catalogue, e.g., the CyberProtect [74] game. These tools are intended for teaching security concepts for people with varied backgrounds and different levels of expertise. They introduce essential security measures by means of exercises in a more or less realistic situation, but give trainees only implicitly a partial security model. There are also more advanced and complex cyber security learning systems, such as CyberNEXS [45] from Leidos (SAIC). The latter is aimed at development of skills on expert level. CyberNEXS is an advanced cybersecurity training system that provides games for four scenarios: network defense, forensics, penetration testing (attacks) and capture the flag. The tools developed in this work provide a complementary approach to war games for teaching cyber security that is based on the analysis and usage of a security model.

4.9 Conclusion

This chapter introduced the concept of rational security design and presented a methodology for optimal long-term planning of security investments. This methodology is based on a multi-level baseline security model and it treats security planning and implementation as a continuous process as opposed to a static state or a one-time investment. A supporting tool—graded security expert system—for modeling security situations and solving optimization problems was implemented using CoCoViLa technology.

The graded security expert system and modeling methodology has two main goals. The first is to provide security managers an efficient approach for planning and optimizing cyber security investments, and the second is to serve as an educational tool for security training. In addition, the tool has been used by third parties as a research platform.

The practical applicability of this methodology depends on the availability of accurate expert data and this is the obstacle for all similar approaches. The availability of tools that can be used to process the data and add value might encourage the collection of high quality input data.

This application has demonstrated the suitability of the CoCoViLa technology for implementing such a tool. The visual interface and visualization including Pareto sets of solutions is usable for security managers. A fast optimizer and interactive user interface support quick experimentation. Extensions developed by third parties show the reusability of the components.

5 Conclusions and Discussion

This chapter concludes the dissertation. It contains a summary of the main results, a discussion and some ideas for future work.

5.1 Main Results

The main results of this work can be summarized in the following three points.

- It was demonstrated that the CoCoViLa platform and its model based software engineering technology can be used in the cyber security field for modeling and simulation. Improvements and extensions to the platform (e.g., production rules, scheme superclasses, rich components, additional specification language syntax, improved user interface, performance improvements, flexible compiling class loader, graceful package evolution, many bug fixes) were developed to better support cyber security simulations.

- The model based software engineering technology was extended with the use of OWL ontologies for developing domain specific visual specification languages. A new concept—DSL meta-model ontology—was introduced and the support of this concept was prototypically implemented in CoCoViLa. This extension facilitates the semantic integration and reuse of existing software artefacts and domain knowledge.

- An application example for analyzing security situations and optimizing cyber security expenses was developed as a CoCoViLa package. This package implements a graded security model based cost optimization approach. It contains a novel and efficient optimizer which employs a discrete dynamic programming algorithm. The software has a simple graphical user interface and a visualization component that is easy to use for security experts. This package has been used for teaching and in pilot projects in banks.

5.2 Discussion

Security is a complex topic and no single solution will mitigate or elliminate all threats or fix all issues. Security is not a state or product. Keeping a system secure is an evolving process that is composed of multiple layers and a variety of measures. The availability of suitable tools, methodologies, and education enables the people responsible for cyber security to better achieve their goals.

In security, generally the weakest link determines the overall security level. This principle must be kept in mind while analyzing security situations using simulation tools. An interactive simulation tool can be used, for example, in decision support in a quickly changing environment for studying multiple scenarios and for simulating several solutions to try out the effect of different values of parameters. A simulation can provide valuable insights and help to discover hidden properties of systems, but it cannot give strong guarantees like a formal verification tool could. Hence, simulation is not a replacement for careful analysis via other means in applications where security matters. This is especially relevant in cyber security where the adversaries may be resourceful, motivated and intelligent, therefore able to discover weak spots in the defense and carefully plan multi-stage attacks.

In this work some universal issues of simulation—model and simulation verification, and the quality of input data—were not discussed as these are separate and complex topics in itself while not being unique to the cyber security domain. However, the technology described here may have the potential to facilitate the semantic integration of existing simulation resources. Open source simulation tools and models are also a part of the solution. The availability of free and open source tools that enable easy integration could motivate studies to gather high quality input data for simulations. Moreover, the quality of models should improve over time as they are reused and gradually refined.

All software developed in the scope of this work should be considered as a proof of concept. Developing a production quality development environment requires a strong product development team and a lot of resources and engineering work. A prototype is, however, a solid starting point for beginning to develop a product.

A distinguishing feature of CoCoViLa is the use of structural synthesis of programs (SSP) which is a form of automatic synthesis of programs based on propositional calculus. SSP uses intuitionistic logic for describing the structure of a program at the level of detail that allows to compose the program automatically from pieces, including, for example, Java methods. This approach gives two main advantages. The first is the flexibility that comes from the ability to solve multiple problems using the same model, hence supporting reuse and increasing the value of models. The second advantage manifests in supporting users in quick development and experimentation. When an expert user knows intuitively the available inputs and desired outputs, the planner is able to figure out some intermediate computation steps given only the specification of the goal. The user does not have to spend time for specifying all intermediate steps, because CoCoViLa helps, within its limits, to overcome the gap between intuition and actual implementation in an automated way. In some situations this capability may, however, become a downside as the user has little control over the path to the solution.

Observations from empirical experience with a limited number of end users suggests that the CoCoViLa technology based cyber security simulation applications have been usable for cyber security experts and students for solving complex simulation problems. Visual specifications, the use of domain specific concepts and the

ability to run simulations without programming is generally well received by end users. Therefore, from the perspective of application engineering, the approach seems to be promising, worth developing further and applying in wider contexts. From the perspective of domain engineering, however, the functionality currently available is not yet satisfying. For example, debugging can be a time consuming challenge and the support provided by the platform for debugging is still quite primitive. Improving this aspect, in addition to several other enhancements, is part of the work planned for future. Some of these ideas are discussed in the following section.

5.3 Future Work

There are several possible directions for future work. Some areas are more of scientific interest, others require software engineering effort to make the technology more usable and robust.

The current version of CoCoViLa is a stand-alone application. This leads to duplication of effort, as there are many features that are needed by any development environment. CoCoViLa functionality should be ported to an existing and well supported integrated development environment (IDE) such as Eclipse [24]. It also relates to the debugging challenges. Integrating CoCoViLa into an IDE might provide some synergy in implementing debugging facilities. Moreover, building and supporting an active user community is necessary for the long term viability of a software project. This may be easier if CoCoViLa was a part of a larger software ecosystem.

CoCoViLa and its applications have been built on the Java platform that has been solid and served its purpose well. The support of other languages, for example Haskell and R, could be helpful in providing more options for developing component libraries.

The future plans also include work related to combining rules represented in Semantic Web Rule Language (SWRL) with domain ontologies. This will allow to model behavioral aspects (e.g., equations) of a domain in DSL meta-model ontology that is currently not supported.

Another area where research is needed is the visual representation of models. Currently there is only one hierarchical visual representation of a model. While hierarchy helps to manage the complexity of large models, there is a limit how much information can be shown in a diagram for it to remain readable. Complex models tend to push this limit. Other modeling tools handle this issue with the support of multiple layers or different views, showing only some aspects of the model at a time. A similar functionality could be useful in CoCoViLa.

Bibliography

[1] Sherly Abraham and Lifang Shih. Towards an Integrative Learning Approach in Cybersecurity Education. In *Proceedings of the 2015 Information Security Curriculum Development Conference*, InfoSec '15, pages 11:1–11:1, New York, NY, USA, 2015. ACM.

[2] Geert Alberghs. Improving the Graded Security Model & -Expert System. Master's thesis, École Supérieure d'Informatique Électronique Automatique, 2011.

[3] Geert Alberghs, Pavel Grigorenko, and Jüri Kivimaa. Quantitative system reliability approach for optimizing IT security costs in an AI environment. In Jaan Penjam, editor, *Proceedings of the 12th Symposium on Programming Languages and Software Tools, SPLST'11*, pages 219–230, Tallinn, Estonia, 5–7 October 2011. TUT Press.

[4] Scott W. Ambler. The Non-Existent Software Crisis: Debunking the Chaos Report. *Dr. Dobb's*, Feb 2014.

[5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[6] Jerry Banks, John Carson, Barry Nelson, and David Nicol. *Discrete-Event System Simulation*. Pearson/Prentice Hall, 4th edition, 2005.

[7] N Md Jubair Basha, Salman Abdul Moiz, and Mohammed Rizwanullah. Model based Software Develeopment: Issues & Challenges. *Special Issue of International Journal of Computer Science & Informatics (IJCSI)*, 2(1), 2012.

[8] Jim Blythe, Aaron Botello, Joseph Sutton, David Mazzocco, Jerry Lin, Marc Spraragen, and Michael Zyda. Testing Cyber Security with Simulated Humans. In *Innovative Applications of Artificial Intelligence*, 2011.

[9] Frederick P. Brooks, Jr. No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20(4):10–19, April 1987.

[10] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational Choice of Security Measures Via Multi-parameter Attack Trees. In

Javier Lopez, editor, *Critical Information Infrastructures Security: First International Workshop, CRITIS 2006, Samos, Greece, August 31–September 1, 2006. Revised Papers*, pages 235–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[11] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer, 2010.

[12] Deniz Çetinkaya, Alexander Verbraeck, and Mamadou D. Seck. Model Continuity in Discrete Event Simulation: A Framework for Model-Driven Development of Simulation Models. *ACM Trans. Model. Comput. Simul.*, 25(3):17:1–17:24, April 2015.

[13] Fred Cohen. Simulating Cyber Attacks, Defenses, and Consequences. `http://all.net/journal/ntb/simulate/simulate.html`, March 1999.

[14] The Common Criteria. Common Criteria for Information Technology Security Evaluation. `http://www.commoncriteriaportal.org/` (10 May 2016).

[15] James B. Dabney and Thomas L. Harman. *Mastering SIMULINK*. Prentice Hall, 2001.

[16] Innovation & Skills Department for Business. Information security breaches survey 2015. `https://www.gov.uk/government/publications/information-security-breaches-survey-2015`.

[17] CoCoViLa development group. CoCoViLa documentation. `http://cocovila.github.io/documentation.html`.

[18] CoCoViLa development group. CoCoViLa specification language. `https://github.com/CoCoViLa/CoCoViLa/wiki/Specification-language`, 2015.

[19] Estonian Information System Authority. Three-level IT baseline security system ISKE. `https://www.ria.ee/iske-en`.

[20] J. Laurenz Eveleens and Chris Verhoef. The rise and fall of the Chaos report figures. *IEEE Software*, 27(1):30–36, Jan 2010.

[21] Jose Evora, Jose Juan Hernandez, and Mario Hernandez. Advantages of Model Driven Engineering for studying complex systems. *Natural Computing*, 14(1):129–144, 2014.

[22] Stanford Center for Biomedical Informatics Research (BMIR). Protégé. `http://protege.stanford.edu/`, 2016.

[23] International Organization for Standardization. Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418, Dec 2010.

[24] The Eclipse Foundation. Eclipse desktop & web IDEs. `http://www.eclipse.org/ide/`.

[25] Robert France and Bernhard Rumpe. Model engineering. *Software and Systems Modeling*, 2(2):73–75, 2003.

[26] Jose M. Garrido and Tridib Bandyopadhyay. Simulation Model Development in Information Security Education. In *2009 Information Security Curriculum Development Conference*, InfoSecCD '09, pages 21–26, New York, NY, USA, 2009. ACM.

[27] German Federal Office for Information Security (BSI). BSI-Standard 100-2 IT Grundschutz Methodology. `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/BSIStandards/standard_100-2_e_pdf.pdf`.

[28] Lawrence A. Gordon and Martin P. Loeb. *Managing Cybersecurity Resources: A Cost-Benefit Analysis*. McGraw-Hill Education, 2006.

[29] Pavel Grigorenko. *Higher-Order Attribute Semantics of Flat Languages*. PhD thesis, Tallinn University of Technology, 2010.

[30] Pavel Grigorenko. *Software user manual for a lightweight and modular expert system shell for the usage in decision support system, version 1.7*. Institute of Cybernetics at TUT, 2011. `http://cocovila.github.io/files/documentation/ExpSys_UserDoc_1.7.pdf`.

[31] Pavel Grigorenko, Ando Saabas, and Enn Tyugu. COCOVILA - Compiler-Compiler for Visual Languages. *Electr. Notes Theor. Comput. Sci.*, 141(4):137–142, 2005.

[32] Pavel Grigorenko and Enn Tyugu. Deep semantics of visual languages. In E. Tyugu and T. Yamaguchi, editors, *Proceedings of the Seventh Joint Conference on Knowledge-based software engineering*, volume 140 of *Frontiers in Artificial Intelligence and Applications*, pages 83–95. IOS Press, 2006.

[33] Gunnar Grossschmidt and Mait Harf. Simulation of an electro-hydraulic servo-valve in NUT programming environment. In Norbert Giambiasi and Claudia Frydman, editors, *Simulation in Industry '2001: 13th European Simulation Symposium 2001 ESS'2001, October 18–20, 2001, Marseille, France*, pages 229–233. Ghent, Belgium: SCS Europe, 2001.

[34] Gunnar Grossschmidt and Mait Harf. COCO-SIM – object-oriented multi-pole modelling and simulation environment for fluid power systems. Part 2: Modelling and simulation of hydraulic-mechanical load-sensing system. *International Journal of Fluid Power*, 10(3):71–85, 2009.

[35] Gunnar Grossschmidt and Mait Harf. Multi-pole modeling and simulation of an electro-hydraulic servo-system in an intelligent programming environment. *International Journal of Fluid Power*, 17(1):1–13, 2016.

[36] Standish Group. CHAOS Report 2015. Report, Standish Group, 2015.

[37] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013. Available at `http://www.w3.org/TR/sparql11-query/`.

[38] Marko Jahnke, Gabriel Klein, Jens Tölle, and Peter Martini. Protecting Military Networks with GrADAR - An Approach for Graph-based Automated Denial-of-Service Attack Response. In *Proceedings of the International Military Communication Conference (MCC 2009)*, Prague, Czech Republic, September 2009.

[39] Andres Järv. Integrating IT security system ISKE with the Graded Security Model to enhance information systems audit prioritization. Master's thesis, Tallinn University of Technology, 2013.

[40] Toomas Kirt and Jüri Kivimaa. Optimizing IT security costs by evolutionary algorithms. In Christian Czosseck and Karlis Podins, editors, *Conference on Cyber Conflict Proceedings*, pages 145–160, Tallinn, Estonia, 15–18 June 2010. Cooperative Cyber Defence Centre of Excellence Publications.

[41] Jüri Kivimaa. *A Cost Optimizing Model for IT Security*. PhD thesis, Estonian Business School, 2013.

[42] Jüri Kivimaa and Toomas Kirt. Evolutionary Algorithms for Optimal Selection of Security Measures. In Rain Ottis, editor, *Proceedings of the 10th European Conferences on Information Warfare and Security, ECIW'11*, pages 172–184, Tallinn, Estonia, 7–8 July 2011. Academic Publishers.

[43] Alexander Kott. *Network Science and Cybersecurity*, chapter Towards Fundamental Science of Cyber Security, pages 1–13. Springer New York, New York, NY, 2014.

[44] Sylvain P. Leblanc, Andrew Partington, Ian Chapman, and Mélanie Bernier. An Overview of Cyber Attack and Computer Network Operations Simulation. In *Proceedings of the 2011 Military Modeling & Simulation Symposium*, MMS '11, pages 92–100, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[45] Leidos. CyberNEXS Cyber Security Training. https://www.leidos.com/cybersecurity/solutions/CyberNEXS.

[46] Xiaojun Liu, Yuhong Xiong, and Edward A. Lee. The Ptolemy II Framework for Visual Languages. In *HCC*, 2001.

[47] Georg Lobsenz. DOE Adopts New "Graded" Terrorist Protection Policy, 2008. http://pogoarchives.org/m/nss/energydaily-20080826.pdf.

[48] Riina Maigre. *Composition of web services on large service models*. PhD thesis, Institute of Cybernetics at Tallinn University of Technology, 2011.

[49] Riina Maigre, Peep Küngas, Mihhail Matskin, and Enn Tyugu. Handling Large Web Services Models in a Federated Governmental Information System. In *Internet and Web Applications and Services, 2008. ICIW '08. Third International Conference on*, pages 626–631, June 2008.

[50] Grigori Mints and Enn Tyugu. The Programming System PRIZ. In *Baltic Computer Science*, pages 1–17, 1991.

[51] National Infrastructure against Cybercrime (NICC). *Process Control Security in the Cybercrime Information Exchange NICC*. TNO; Nationale Infrastructuur ter bestrijding van Cybercrime (NIC), 2009.

[52] Peter Naur and Brian Randell, editors. *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 Oct. 1968, Brussels*. NATO Scientific Affairs Division, 1969.

[53] Anthony G. Oettinger. The Hardware-software Complementarity. *Commun. ACM*, 10(10):604–606, October 1967.

[54] U. S. Department of Defense. National Industrial Security Program Operating Manual (NISPOM). 2006.

[55] Andres Ojamaa and Enn Tyugu. Rich Components of Extendable Simulation Platform. In Hamid R. Arabnia, editor, *MSV*, pages 121–127. CSREA Press, 2007.

[56] Alessandro Oltramari, Lorrie Faith Cranor, Robert J. Walls, and Patrick Drew McDaniel. Building an Ontology of Cyber Security. In Kathryn Blackmond Laskey, Ian Emmons, and Paulo C. G. Costa, editors, *Proceedings of the Ninth Conference on Semantic Technology for Intelligence, Defense, and Security, Fairfax VA, USA, November 18-21, 2014.*, volume 1304 of *CEUR Workshop Proceedings*, pages 54–61. CEUR-WS.org, 2014.

[57] OPNET Technologies, Inc. OPNET Solutions. OPNET Modeler. http://www.opnet.com/.

[58] Min Ouyang. Review on modeling and simulation of interdependent critical infrastructure systems. *Reliability Engineering & System Safety*, 121:43–60, 2014.

[59] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[60] Stylianos Papanastasiou, Jens Mittag, Erik G Ström, and Hannes Hartenstein. Bridging the Gap Between Physical Layer Emulation and Network Simulation. In *Proceedings of IEEE WCNC Conference, 2010*, April 2010.

[61] Ike Patterson, James Nutaro, Glenn Allgood, Teja Kuruganti, and David Fugate. Optimizing Investments in Cyber-security for Critical Infrastructure. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 20:1–20:4, New York, NY, USA, 2013. ACM.

[62] Kai Petersen. Measuring and predicting software productivity: A systematic map and review. *Information and Software Technology*, 53(4):317 – 343, 2011. Special section: Software Engineering track of the 24th Annual Symposium on Applied ComputingSoftware Engineering track of the 24th Annual Symposium on Applied Computing.

[63] Alireza Shameli-Sendi, Rouzbeh Aghababaei-Barzegar, and Mohamed Cheriet. Taxonomy of information security risk assessment (ISRA). *Computers & Security*, 57:14–30, 2016.

[64] Matti Tedre. *The Science of Computing: Shaping a Discipline*. Chapman & Hall/CRC, 2014.

[65] Michael Thompson and Cynthia Irvine. Active Learning with the Cyber-CIEGE Video Game. In *Proceedings of the 4th Conference on Cyber Security Experimentation and Test*, CSET'11, Berkeley, CA, USA, 2011. USENIX Association.

[66] Juha-Pekka Tolvanen and Steven Kelly. MetaEdit+: defining and using integrated domain-specific modeling languages. In Shail Arora and Gary T. Leavens, editors, *OOPSLA Companion*, pages 819–820. ACM, 2009.

[67] Federico Tomassetti, Marco Torchiano, Alessandro Tiso, Filippo Ricca, and Gianna Reggio. Maturity of software modelling and model driven engineering: A survey in the Italian industry. In *Evaluation Assessment in Software Engineering (EASE 2012), 16th International Conference on*, pages 91–100, May 2012.

[68] Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. Relevance, benefits, and problems of software modelling and model driven techniques—A survey in the Italian industry. *Journal of Systems and Software*, 86(8):2110 – 2126, 2013.

[69] Thomas Toth and Christopher Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *Proceedings of the 18th Computer Security Applications Conference 2002*, pages 301–310, 2002.

[70] Enn Tyugu. The structural synthesis of programs. *Algorithms in Modern Mathematics and Computer Science*, pages 290–303, 1979.

[71] Enn Tyugu. *Algorithms and Architectures of Artificial Intelligence*, volume 159 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2007.

[72] Enn Tyugu. Command and Control of Cyber Weapons. In Christian Czosseck, Rain Ottis, and Katharina Ziolkowski, editors, *Proceedings of 4th International Conference on Cyber Conflict (CYCON)*, pages 333–343, Tallinn, Estonia, 5–8 June 2012. CCD COE Publications.

[73] Enn Tyugu, Mihhail Matskin, and Jaan Penjam. Applications of Structural Synthesis of Programs. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I*, pages 551–569, London, UK, 1999. Springer-Verlag.

[74] U.S. DoD, Defense Information Systems Agency. CyberProtect, version 1.1. `http://iase.disa.mil/eta/`.

[75] U.S. DoE, Office of Information Resources. DOE O 470.3B, Graded Security Protection (GSP) Policy. `https://www.directives.doe.gov/directives/0470.3-BOrder-b/view`.

[76] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *SimuTools*, page 60, 2008.

[77] Verizon. Data breach digest. Scenarios from the field. Technical report, Verizon, 2016.

[78] Rossouw von Solms and Johan van Niekerk. From information security to cyber security. *Computers & Security*, 38:97–102, 2013. Cybercrime in the Digital Economy.

[79] Eric Winsberg. Computer Simulations in Science. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Stanford University, summer 2015 edition, 2015.

[80] Charles Cresson Wood. Why information security is now multi-disciplinary, multi-departmental, and multi-organizational in nature. *Computer Fraud & Security*, 2004(1):16–17, 2004.

[81] Peter Wood. The hacker's top five routes into the network (and how to block them). *Network Security*, 2006(2):5–9, 2006.

[82] Yong-liu and Aiguang-yang. Research and Application of Software-reuse. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, volume 3, pages 588–593, July 2007.

Acknowledgments

I am grateful to my supervisors Enn Tõugu and Jaan Penjam for their guidance and support.

I thank my colleagues at the Institute of Cybernetics—Pavel Grigorenko, Hele-Mai Haav, Mait Harf, Vahur Kotkas, Riina Maigre—for many fruitful discussions and their insightful ideas and comments. I am also grateful to Jüri Kivimaa and Gabriel Klein for a successful and gratifying collaboration. I thank Riina Maigre, Hele-Mai Haav and Eve Kann for proofreading parts of the manuscript and providing valuable feedback.

Thanks to the Estonian Information Technology Foundation, the Tiger University programme and the following projects for financial support of this work: ERDF funded ICT national programme project MBJSDT (3.2.1201.13-0026), ESF funded Estonian Doctoral School in ICT (1.2.0401.09-0081), Estonian Ministry of Education and Research target-financed themes 0322709s06, 0140007s12 and institutional research grant IUT33-13, and Estonian Ministry of Defense contracts 508/0711, 372/0807.

Software Technology for Cyber Security Simulations

Abstract

Cyber security is a broad and multifaceted field that includes aspects related to the security of IT systems starting from low level technical details and practical software engineering problems to economic and even political considerations.

This dissertation studies the nature of the cyber security field from the perspective of developing computer simulations. New approaches are sought to improve the state of the art of software engineering practices and simulation tools with the goal to better support cyber security specialists in solving complex problems.

The complexity of the problems in the cyber security field demands high flexibility and adaptability from the technologies used for simulation. The software tools have to be multifunctional, extensible and easy to integrate with existing IT infrastructure. The technology has to be usable by security specialists and analysts who are not programmers and want to focus on solving problems, not writing code.

A model based software engineering technology and a supporting simulation platform called CoCoViLa have been developed at the Institute of Cybernetics at TUT. This technology and the simulation platform have been used successfully for applications in the mechanical engineering field. In the context of the work done for this dissertation, this approach was experimentally applied to problems from the cyber security domain to demonstrate broader applicability of the CoCoViLa technology.

As an application example a graded security based cost optimization method was developed and implemented as software package. This package includes a novel optimizer that helps to select the most efficient security measures for securing an information system taking into account a set of prescribed security requirements and a limited budget. It also helps to plan the implementation of the measures and divide the costs of the security measures over a longer period spanning several years in an optimal way.

Various experimental enhancements of the simulation platform and technology were proposed and evaluated while developing applications related to the context of this work. As one of the most significant outcomes, the existing model based software engineering technology was extended with the support of using OWL ontologies for developing domain specific visual specification languages. This extension helps achieving the correctness of specifications, facilitates the reuse of existing software artefacts and supports the semantic integration of applications from different problem domains.

Tarkvaratehnika küberturbe simulatsioonide jaoks

Kokkuvõte

Küberturve on lai ja mitmekülgne valdkond, mis hõlmab IT-süsteemidega seotud aspekte alates madala taseme tehnilistest üksikasjadest ja praktilistest tarkvaraarenduse küsimustest kuni majanduslike ja isegi poliitiliste kaalutlusteni.

Käesolevas väitekirjas uuritakse küberturbe valdkonna eripära arvutisimulatsioonide realiseerimise seisukohast ning otsitakse võimalusi simuleerimisvahendite tarkvaratehnika edasiarendamiseks eesmärgiga toetada küberturbe spetsialistide tööd kompleksete ülesannete lahendamisel.

Küberturbe valdkonna ja ülesannete komplekssus nõuab simulatsioone toetavatelt tehnikatelt paindlikkust ja kohandatavust. Tarkvaralised tööriistad peavad olema multifunktsionaalsed, laiendatavad ning kergesti lõimitavad olemasoleva infotehnoloogilise taristuga. Tehnika arendamisel tuleb arvestada asjaoluga, et küberturbe spetsialistid ja analüütikud ei ole tarkvaraarendajad ning nende huvi on eelkõige lahendada oma valdkonna ülesandeid, mitte keskenduda programmeerimisele.

Doktoritöö käigus katsetati TTÜ Küberneetika instituudis välja töötatud ja inseneriarvutuste jaoks edukalt kasutatud simuleerimisplatvormi CoCoViLa rakendatavust küberturbe valdkonnas. CoCoViLa toetab mudelpõhist tarkvaratehnikat, visuaalseid valdkonnaspetsiifilisi spetsifitseerimiskeeli ning automaatset programmide sünteesi. Katsetused näitavad, et CoCoViLa tehnoloogia sobib hästi mitmete küberturbe valdkonna ülesannete lahendamiseks.

Ühe küberturbe valdkonna rakendusnäitena, mida väitekirjas kirjeldatakse, on välja töötatud astmelisel andmeturbe mudelil põhinev küberturbe kulude optimeerimise metoodika ning seda toetav tarkvarapakett. Pakett sisaldab uudset optimeerijat, mis aitab piiratud eelarve tingimustes tagada infosüsteemide turvanõuete võimalikult efektiivse täitmise ning võimaldab kulutusi optimaalselt planeerida mitmeaastasele perioodile.

Doktoritöö jooksul realiseeritud küberturbe simulatsioonide arendamisel otsiti võimalusi olemasoleva tehnoloogia ja platvormi edasiarendamiseks. Selle tulemusel loodi mitmeid küberturbe simulatsioone toetavaid laiendusi. Üks olulistest täiendustest, mida väitekirjas käsitletakse, on uudne viis OWL ontoloogiate rakendamiseks valdkonnaspetsiifiliste visuaalsete spetsifitseerimiskeelte arendamisel, mis aitab tagada spetsifikatsioonide korrektsust, soodustab olemasolevate ressursside paremat taaskasutust ning toetab eri valdkondade rakenduste semantilist lõimimist.

A Listings and Figures

A.1 SPARQL query for finding visual classes

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ccvl:
  <http://www.cs.ioc.ee/cocovila/ontologies/2015/cocovila-system#>

SELECT ?vclass ?image ?icon ?className ?descr
       ?extends ?dc ?relclass ?spectext
WHERE {
    ?vclass rdf:type                    ccvl:VisualClass;
            ccvl:isRepresentedByImage   ?image;
            ccvl:hasIcon                ?icon;
            ccvl:hasVisualClassName     ?className.
    OPTIONAL {
        ?vclass     rdf:type                ?relclass.
        FILTER (?relclass = ccvl:RelationClass)
    }
    OPTIONAL {
        ?javaclass  rdf:type                ccvl:JavaClass;
                    ccvl:hasClassName       ?extends.
        ?vclass     ccvl:isImplementedBy    ?javaclass.
    }
    OPTIONAL {
        ?vclass ccvl:hasSpecText        ?spectext.
    }
    OPTIONAL {
        ?vclass ccvl:hasDescription     ?descr.
    }
    OPTIONAL {
        ?vclass     rdf:type            ?dc.
        FILTER NOT EXISTS {
            ?vclass   rdf:type          ?type.
            ?type     rdfs:subClassOf   ?dc.
            FILTER NOT EXISTS {
                ?type owl:equivalentClass ?dc.
            }
        }
        FILTER (
            !bound(?dc) || (isIRI(?dc)
                && !strstarts(str(?dc),
        "http://www.cs.ioc.ee/cocovila/ontologies/2015/cocovila-system#")
                && ?dc != owl:NamedIndividual)
        )
    }
}
```

A.2 Listing of toolbox ontology in OWL functional syntax

```
1  Prefix(:=<http://www.cs.ioc.ee/cocovila/ontologies/2015/toolbox#>)
2  Prefix(tb:=<http://www.cs.ioc.ee/cocovila/ontologies/2015/toolbox#>)
3  Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
4  Prefix(ccvl:=<http://www.cs.ioc.ee/cocovila/ontologies/2015/cocovila-
       system#>)
5
6  Ontology(<http://www.cs.ioc.ee/cocovila/ontologies/2015/toolbox>
7  Import(<http://www.cs.ioc.ee/cocovila/ontologies/2015/cocovila-system>)
8
9  Declaration(NamedIndividual(tb:fieldClockInitState))
10 Declaration(NamedIndividual(tb:fieldClockPeriod))
11 Declaration(NamedIndividual(tb:fieldGraphSeriesName))
12 Declaration(NamedIndividual(tb:fieldProcessNumSteps))
13 Declaration(NamedIndividual(tb:jcObject))
14 Declaration(NamedIndividual(tb:portClockTime))
15 Declaration(NamedIndividual(tb:portGraphX))
16 Declaration(NamedIndividual(tb:portGraphY))
17 Declaration(NamedIndividual(tb:vcClock))
18 Declaration(NamedIndividual(tb:vcGraph))
19 Declaration(NamedIndividual(tb:vcProcess))
20
21 DataPropertyAssertion(ccvl:hasFieldName tb:fieldClockInitState "initstate
       ")
22 DataPropertyAssertion(ccvl:hasFieldType tb:fieldClockInitState "double")
23
24 DataPropertyAssertion(ccvl:hasFieldName tb:fieldClockPeriod "period")
25 DataPropertyAssertion(ccvl:hasFieldType tb:fieldClockPeriod "double")
26
27 DataPropertyAssertion(ccvl:hasFieldName tb:fieldGraphSeriesName "
       seriesName")
28 DataPropertyAssertion(ccvl:hasFieldType tb:fieldGraphSeriesName "String")
29
30 DataPropertyAssertion(ccvl:hasFieldName tb:fieldProcessNumSteps "numSteps
       ")
31 DataPropertyAssertion(ccvl:hasFieldType tb:fieldProcessNumSteps "int")
32
33 DataPropertyAssertion(ccvl:hasClassName tb:jcObject "Object")
34
35 DataPropertyAssertion(ccvl:hasPortName tb:portClockTime "state")
36 DataPropertyAssertion(ccvl:hasPortOffsetX tb:portClockTime "50"^^xsd:
       integer)
37 DataPropertyAssertion(ccvl:hasPortOffsetY tb:portClockTime "2"^^xsd:
       integer)
38 DataPropertyAssertion(ccvl:hasPortType tb:portClockTime "double")
39
40 DataPropertyAssertion(ccvl:hasPortName tb:portGraphX "x")
41 DataPropertyAssertion(ccvl:hasPortOffsetX tb:portGraphX "50"^^xsd:integer)
42 DataPropertyAssertion(ccvl:hasPortOffsetY tb:portGraphX "86"^^xsd:integer)
43 DataPropertyAssertion(ccvl:hasPortType tb:portGraphX "double")
44
45 DataPropertyAssertion(ccvl:hasPortName tb:portGraphY "y")
46 DataPropertyAssertion(ccvl:hasPortOffsetX tb:portGraphY "14"^^xsd:integer)
47 DataPropertyAssertion(ccvl:hasPortOffsetY tb:portGraphY "50"^^xsd:integer)
48 DataPropertyAssertion(ccvl:hasPortType tb:portGraphY "double")
49
50 ObjectPropertyAssertion(ccvl:hasField tb:vcClock tb:fieldClockInitState)
51 ObjectPropertyAssertion(ccvl:hasField tb:vcClock tb:fieldClockPeriod)
52 ObjectPropertyAssertion(ccvl:hasPort tb:vcClock tb:portClockTime)
```

```
53 ObjectPropertyAssertion(ccvl:isImplementedBy tb:vcClock tb:jcObject)
54 DataPropertyAssertion(ccvl:hasDescription tb:vcClock "Basic clock"^^xsd:
       string)
55 DataPropertyAssertion(ccvl:hasIcon tb:vcClock "clock.png"^^xsd:anyURI)
56 DataPropertyAssertion(ccvl:hasSpecText tb:vcClock "double period;
57 double initstate, state, nextstate, finalstate;
58 nextstate = state + period;
59 state = finalstate;")
60 DataPropertyAssertion(ccvl:hasVisualClassName tb:vcClock "Clock")
61 DataPropertyAssertion(ccvl:isRepresentedByImage tb:vcClock "clock.svg")
62
63 ObjectPropertyAssertion(ccvl:hasField tb:vcGraph tb:fieldGraphSeriesName)
64 ObjectPropertyAssertion(ccvl:hasPort tb:vcGraph tb:portGraphX)
65 ObjectPropertyAssertion(ccvl:hasPort tb:vcGraph tb:portGraphY)
66 DataPropertyAssertion(ccvl:hasDescription tb:vcGraph "Single series graph
       "^^xsd:string)
67 DataPropertyAssertion(ccvl:hasIcon tb:vcGraph "SingleSeriesGraph.png"^^xsd
       :anyURI)
68 DataPropertyAssertion(ccvl:hasVisualClassName tb:vcGraph "
       SingleSeriesGraph")
69 DataPropertyAssertion(ccvl:isRepresentedByImage tb:vcGraph "graph.svg")
70
71 ObjectPropertyAssertion(ccvl:hasField tb:vcProcess tb:fieldProcessNumSteps
       )
72 DataPropertyAssertion(ccvl:hasDescription tb:vcProcess "Simulation engine
       "^^xsd:string)
73 DataPropertyAssertion(ccvl:hasIcon tb:vcProcess "proc-icon.png"^^xsd:
       anyURI)
74 DataPropertyAssertion(ccvl:hasVisualClassName tb:vcProcess "Proc")
75 DataPropertyAssertion(ccvl:isRepresentedByImage tb:vcProcess "proc.svg")
76 )
```

A.3 Visualization of attack tree simulation DSL meta-model ontology

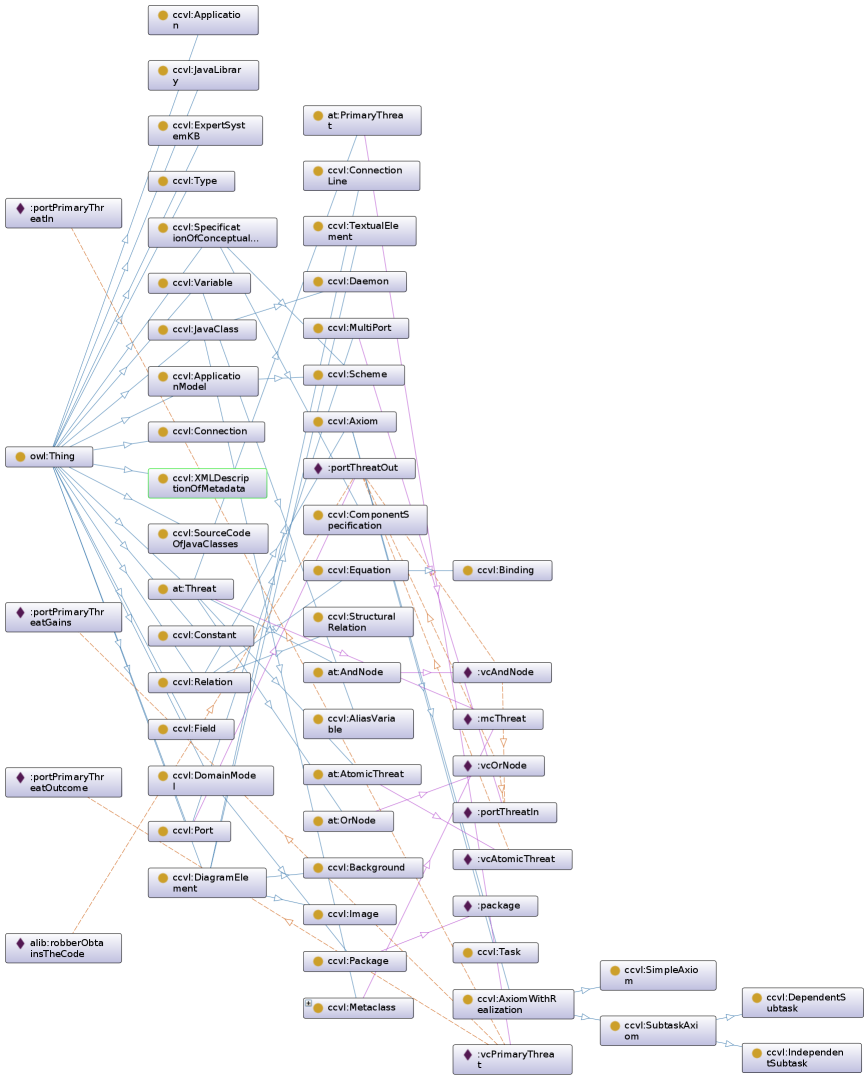


Figure A.1: DSL meta-model ontology

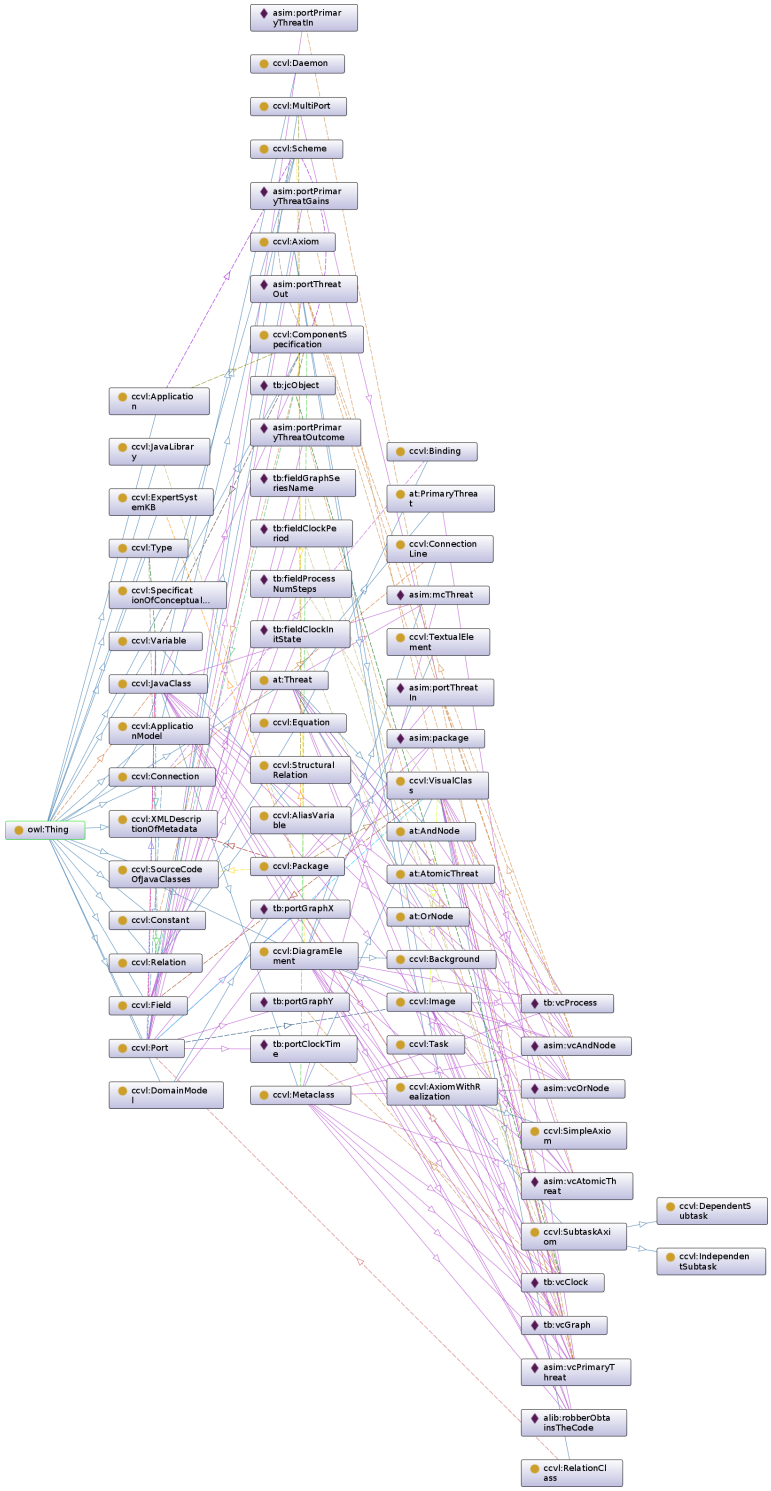


Figure A.2: DSL meta-model ontology with inferred axioms

Publication I

Kotkas, Vahur; Ojamaa, Andres; Grigorenko, Pavel; Maigre, Riina; Harf, Mait; Tyugu, Enn (2011).

CoCoViLa as a Multifunctional Simulation Platform

Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques: 21–25 March 2011, Barcelona, Spain, SIMUTools 2011. Brussels: ICST, 195–205.

CoCoViLa as a Multifunctional Simulation Platform

Vahur Kotkas Andres Ojamaa Pavel Grigorenko
Riina Maigre Mait Harf Enn Tyugu

Institute of Cybernetics at Tallinn University of Technology
Akadeemia tee 21, 12618 Tallinn, Estonia
{vahur, andres.ojamaa, pavelg, riina, mait, tyugu}@cs.ioc.ee

ABSTRACT

A flexible Java-based simulation platform that includes both continuous-time and discrete event simulation engines and is intended for applications in a variety of domains is presented. The platform supports visual and model-based software development and uses structural synthesis of programs for translating declarative specifications of simulation problems into executable code. Rich components are an important concept of the work. They are implemented as Java classes with additional specifications for program synthesis, and include visual representations as well as daemons supporting continuous interaction with the user during the simulation. The platform is developed as an open-source software, and its extensions can be written in Java and included into simulation packages.

1. INTRODUCTION

We expect that simulation problems are becoming not only computationally heavier, but also considerably more complex in the sense that a single problem may require orchestrated usage of numerous simulation engines, optimization programs as well as visualization and statistics software. A simulation engine and other useful tools are usually built into the simulation platform. It is a common practice that a simulation problem itself is described using a model-based technology, mainly by specifying a model of the simulated system. In the present work we propose to use the model-based approach for constructing and compiling a completely new program for every simulation, using simulation engines, optimizers, visualizers etc. as preprogrammed components. In this case a simulation platform is a model-based software development environment plus assets (components, language, models) for specifying simulation problems. This requires good automation of program construction from a given model.

We have implemented CoCoViLa [4] as a suitable software development environment. It has visual tools, but most importantly, it supports full automatic program construction from specifications that are given visually. The CoCoViLa itself has been developed bearing in mind programming of simulation problems. However, it has been used also in a more general model-based software development, e.g., for composing web services on large service models [11]. In this sense it is similar to visual software development tools like MetaEdit [17]. In our design and implementation, we have paid special attention at flexibility. This allows us to call the CoCoViLa platform multifunctional. In particular, it is applicable for very different specialized problems as almost every aspect (optimization algorithms, simulation engines, etc.) of the simulation can be customized by replacing a component with another from the toolbox or implementing a new component.

The present paper is structured as follows. Section 2 describes the conceptual design of the simulation platform. The specification language is described in Section 3. Section 4 presents implementation aspects and Section 5 contains application examples.

2. DESIGN PRINCIPLES

Our main design decision is that we rely on a completely automatic program construction from a specification, and use a fast synthesis method that gives out a source code ready for compilation and execution. This program synthesis—structural synthesis of programs (SSP) [18] is not new, it has been implemented and used in several earlier software tools [12, 19]. It uses essentially dataflow for composing a program, like, for instance, Simulink [2] does. However, the usage of components in the form of higher-order functions that take synthesized parts of a program as inputs makes a significant difference over conventional dataflow techniques. The program synthesis process uses dataflow recursively for solving so called subtasks—generating inputs for the higher-order functions, i.e., for achieving the goals like "synthesize a body for the loop in this particular component". This makes the method universally applicable – theoretically any algorithm can be synthesized in this way from a suitable specification and a fixed set of preprogrammed components [12]. A program is synthesized piecewise and the pieces are bound together by preprogrammed higher-order functions that realize required control structures. The planner is CoCoViLa's core part responsible for synthesis, it is described in more detail in Section 4.2.

The second design decision is using full capabilities of model-based software development—developing a completes model for each simulation problem that includes not only description of a simulated system, but describes also the usage

SIMUTOOLS 2011, March 21-25, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245553

of simulation engines and other tools—visualizers, optimizers etc. needed for the particular problem. A model-based software development consists of two stages: domain engineering that provides assets for developing applications, and application engineering that uses the assets. Assets of our model-based approach are rich components [13]—Java classes extended with specifications for program synthesis and supplied also with visual representation like Java beans. A rich component may have another class associated with it—that specifies a daemon—a thread that runs permanently and supports user interaction during the problem description and simulation phases. A component with a daemon can behave as an agent and can be compared with agents in simulation environment, for instance, provided by AgentSheets [16].

A collection of rich components for a problem domain constitute a package that is an implementation of a domain specific language (DSL) for this particular domain. Naturally, packages can be restructured using export/import commands, and components of different packages can be used for specifying a simulation problem. Generally applicable components like simulation engines, optimizers, input-output components are collected in a package called toolbox.

We have chosen Java as the implementation environment of our simulation platform. This is justified by useful properties of Java: good portability and interoperability, support for distributed computing, open source ideology, dynamic compilation and loading of classes. Our platform has been developed in a way that does not restrict the usage of Java for programming of classes. And from the other side—all Java types and classes can be used as types of objects in specifications.

3. LANGUAGE

We have implemented one specification language for describing both components and simulation problems. This gives immediately a possibility of developing hierarchical descriptions—a part of a model of simulated system can easily be declared to be a component. This language is built on top of Java and merged with Java in a simple way: a specification is always a comment of a special form included in a Java class. It may happen that a class has nothing else than a specification in it. An important decision has been to keep separated the namespaces of a procedural part written in Java and of a specification inside a Java class. Here is an example of a specification that shows also the form of a specification, and the possibility of usage of equations in a specification:

```
class Complex {/*@
      specification Complex {
         double re, im, arg, mod;
         mod^2 = re^2 + im^2;
         mod * sin(arg) = im;
      } @*/
}
```

The only connection between the actual Java code and the specification is through the method names that refer to implementations of functions in a specification. The following is an example of usage of a Java method (`getMaxVal`) as an implementation of a higher-order function in a specification:

```
class Max {
    /*@ specification Max {
        int arg, val, maxval;
        [arg->val]->maxval{getMaxVal};
    } @*/

    public int getMaxVal(Subtask sbt) {
        ...
        return maxval;
    }
}
```

Argument `sbt` of the method `getMaxVal` should get a synthesized method that implements the interface `Subtask`.

Selection of program synthesis puts restrictions on the specification language: each function usable in synthesis must have a specification describing its input and output conditions. We call these specifications *axioms*, as it has been the convention for structural synthesis of programs. These axioms have a precise logical meaning, but we are able to explain them in terms of dataflow only. The example above includes the axiom

```
[arg->val]->maxval{getMaxVal};
```

This axiom has one input `[arg->val]` that is a subtask describing a function for computing `val` from given `arg` (this function has to be synthesized by the planner). The axiom has an output `maxval`.

The platform supports a textual and a visual representation of the specification language. The specification language has a simple and a rather conventional syntax of declarative compositional languages. It enables one to specify typed objects and bind them with each other by connecting their attributes by equalities. Numeric variables can be bound also by algebraic equations. Constant values can be assigned to variables of any type, as soon as the value has a textual representation. The textual specification enables one also to specify the usage of methods of the class where the specification is included by writing axioms about their applicability, i.e., by giving their pre- and postconditions. Extensibility of the language is achieved by introduction of new types. The following is core of the language:

1. declaration of a component:

   type id;

   This declaration specifies a component of a model with given type and name.

2. binding:

   $var1.portA = var2.portB$;

   This statement specifies an equality between variables (ports) of components.

3. valuation:

   $var1.portC = value$;

   This statement defines a functional dependency with no inputs and with one output that receives a constant value.

4. axiom

   $precondition \rightarrow postcondition\{implementation\}$;

   The precondition of axioms is a list of component names and subtasks. The postcondition is a component name.

The names in precondition show components that are inputs and the name in postcondition shows a component in output of the computation by the function given by the method with the name implementation. A subtask has the form $[x_1, \ldots, x_n \rightarrow y_1, \ldots, y_m]$ and defines a function with inputs and outputs given on the left and right side of the arrow. The function defined by a subtask has to be synthesized and given as an input to the function described by the axiom.

5. equation

   $AExpression = AExpression;$

   Equation defines one or more functional dependencies that are solving functions for variables bound by the equations. Arithmetic expressions are the Java ones, and can be solved only for the variables that have one occurrence in an expression.

6. tuple

   $alias\ id = (ListOfNames);$

   where $ListOfNames$ can include names with wildcards of the form $*.id$. In this case all ports of components of a specification that have the name id are included in $ListOfNames$. For example, `alias state = (*.state)` describes a state vector consisting of states of components.

The visual language describes schemes and, strictly speaking, uses only the first two kinds of statements. However, through pop-up windows one can add also valuations, aliases and equations to a scheme.

4. IMPLEMENTATION

4.1 The platform

From a user's perspective, CoCoViLa consists of two visual editors: the Class Editor for creating simulation packages by developing domain-specific concepts and the Scheme Editor for the visual composition of simulation problems and their execution.

4.1.1 Class Editor

In the Class Editor users can define visual aspects of rich components using drawing capabilities or by importing corresponding bitmaps. Figure 1 shows the development of a component responsible for plotting charts in the window of the Class Editor. The image of the chart contains two ports (dots) for providing data to the axis. A smaller pop-up window is for defining properties of a highlighted port. Another window is for specifying attributes of the given component, e.g., class name, toolbar icon, description and a set of fields of visual interface with types and default values. Functional properties of this component are implemented in a Java class.

4.1.2 Scheme Editor

The Scheme Editor is a multi-purpose tool. It allows to load a simulation package created in the Class Editor (user interface with toolbars and menus is automatically generated from the package description) and to compose visually a simulation problem in a scheme. Schemes can be saved and used as components in other schemes higher in the hierarchy. This is due to the fact that each scheme is a Java

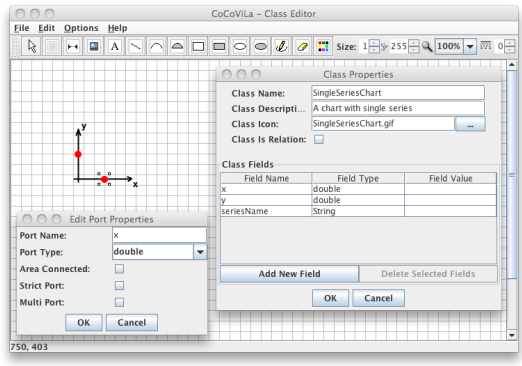


Figure 1: The Class Editor window

class. From a visual description of a problem, a simulation program is synthesized automatically. There are also debugging capabilities (algorithm visualizer, viewer for synthesized code, etc.). An executed simulation problem can show results both in a separate window or display the feedback directly on a scheme.

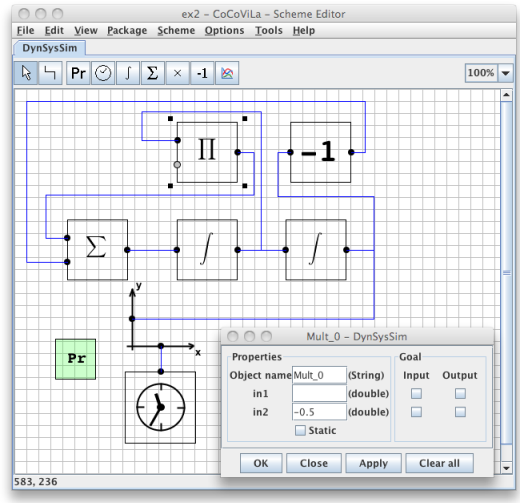


Figure 2: The Scheme Editor window

Figure 2 shows the Scheme Editor with a loaded package for simulating simple dynamic systems. Such systems are described by ordinary differential equations. The scheme shown in the figure has been composed by connecting ports of components of the following types: `Integrator`, `Adder`, `Multiplier`, `Inverter`, `Clock` and `Chart`. A continuous-time simulation engine (`Pr`) is added as a superclass of the class `ex2` of the scheme. The toolbar at the top of the scheme is for adding objects and connections to the scheme. A pop-up window is for specifying attributes of a selected `Multiplier` object. The Scheme Editor is syntax directed

and the correctness of the scheme is forced during editing.

4.2 Planner

The *planner* is a core part of our platform. Its purpose is to transform declarative specifications of simulation problems into executable programs. The planner determines computational paths from initial variables to required goal variables (i.e., tries to solve a given *computational problem* "find values of V from given values of U", where U and V are sets of input and output variables). The planner's task is not only to construct a linear dataflow, but also to solve subtasks (higher-order dataflow) and to perform optimization of an algorithm. Generation of a resulting program's code from an algorithm is then a straightforward process.

Let us have a small example of dataflow planning where a goal is to compute a next state of a simulation from a current state. That is, a computational problem can be expressed with a following statement:

```
state -> nextstate;            (1)
```

The problem can be solved only if a relation between `state` and `nextstate` is specified. Assume the state is just a numerical value and the next state is an increment of a current state by a step which is obtained using a method `getStep`:

```
nextstate = state + step;
-> step{getStep};
out = nextstate;
```

In this case the planner produces the following dataflow if a value of the state is given as input:

```
state = getStep();
nextstate = state + step;
```

Note that the calculation of a variable `out` is not included in the dataflow because it is not required for solving a given problem (1). We can add another statement into our specification that prints the value of the nextstate

```
nextstate -> printed{print};  (2)
```

The dataflow for (1) will not include (2) because (1) does not include a control variable `printed` in a set of outputs. Another computational problem has to be stated:

```
state -> printed;
```

The example above shows how to construct a linear dataflow, i.e., to compute the value of the next state once. Simulation tasks require to compute states in a loop until some satisfying final state is reached. To specify such task in CoCoViLa, subtasks have to be used. The following statement specifies that a final state can be computed from a given initial state if there exists a function that calculates the next state from a given state.

```
[state -> nextstate], istate -> fstate{proc};
```

To solve a topmost computational problem `istate->fstate`, the subtask `state -> nextstate` must be solved. Having (2), subtask is solvable and higher-order dataflow can be constructed by the planner. The synthesized function `state -> nextstate` is passed as an argument to the method `proc` and this method can iteratively call the function to increment the state as long as it is needed.

Following the described technique, simulation engines that require loops and other control structures are implemented using subtasks and the planner takes care of synthesizing bodies of subtasks. In addition, the planner can be invoked at runtime for generating new programs to solve tasks on models that might have been changed dynamically during the simulation.

In general, the synthesis of an algorithm with subtasks has exponential time complexity with respect to the number of subtasks in a specification, as the solvability of one subtask may depend on the solvability on another subtask or it can be the case that one and the same subtask has to be solved repeatedly in one and the same branch. The solvability search is done on an *and-or* tree where *and*-nodes are axioms and *or*-nodes are subtasks. An implemented algorithm is incremental depth-first search with backtracking and additional heuristics.

The planning algorithm can be explained also in terms of logic, and vice versa—theorems of intuitionistic propositional calculus can be encoded as sets of axioms (axioms can be considered as propositional formulas where arrows denote implications and commas denote conjunctions). To test planner's performance, CoCoViLa has been compared to several well-known theorem provers [6]. Full description and results of the benchmark can be found in [3].

To give some details, a formula $((((A \to B) \to A) \to A) \to B) \to B$ (an intuitionistic analog of classical Peirce's law) was elaborated to a special form where it was possible to include copies of parts of initial formula to create more levels of nestedness. The results are summarized in the Table 1.

	n-th level formula					
Tool	2	4	6	7	10	11
CoCoViLa	<0.01	0.05	36.24	1579.6	–	–
STRIP (ch)	<0.01	0.34	3781.3	–	–	–
STRIP (pr)	34.87	–	–	–	–	–
iLeanCoP	–	–	–	–	–	–
iLeanSeP	–	–	–	–	–	–
LJT	<0.01	0.05	35.15	1572.3	–	–
PITP	0.01	0.05	15.73	343.5	–	–
Gandalf	0.02	0.19	0.53	0.9	7.6	–

Table 1: Proof search time (in seconds) of various theorem provers

4.3 Rich components

Rich components are descriptions of domain-specific concepts that are used in simulation. Rich components collected into packages are the building blocks for computational problems and simulation tasks. Such components are designed to consist of four different parts (views).

The first part is the graphical representation. Having such representation we can build schemes of simulation tasks via visual composition of the components.

The graphical representation can be translated into a textual specification—the second part. For textual specification of components we use the composition language described in Section 3. The automatic composition of simulation programs is supported by the transformation of textual specifications into the logical representation used by attribute evaluation algorithms implemented in the simulation plat-

form.

The third part of a rich component is a *metaclass* [4]—the Java class in our case that may contain methods that are the realizations of dependencies described in textual specification of a rich component. The textual specification included into the metaclass is called a *metainterface*. Metainterfaces appear in metaclasses as comments and are used only by the simulation platform to facilitate automatic composition of simulation programs.

The optional fourth part of a rich component is called a *daemon*. The daemon is a Java class that describes a thread that can be started by a user, if it is needed. Daemons enable one to develop flexible interfaces to simulation programs. In addition, daemons have the reference to the scheme being composed by the user. This gives them full control over the scheme enabling one to develop components that can assist the user in the task of composing a simulation model by displaying dialog windows and arbitrarily complex visual feedback or directly modifying the model. For example, a daemon can be used to enforce some complex syntax rules on the structure of the visual scheme.

Simulation packages may contain simulation engines implemented as rich components, in this case such components can be of considerable size. From the other side, rich components may be very small entities that are used intensively in an internal loop of a simulation program. In other words, depending on the implementation and the purpose of the application, rich components can be very lightweight containing only visual and metainterface part or heavyweight including all four functional parts.

4.4 Toolbox

Toolbox is a package in CoCoViLa that contains a number of components useful for building simulation programs. It includes several simulation engines and also visualization components. Another feature implemented as a standalone tool is an expert system environment (some simulation problems require handling of the expert knowledge).

4.4.1 Simulation engines

The toolbox includes basic simulation engines as components for driving discrete-event, continuous time and hybrid simulations. Simulation engines are used as superclasses of schemes, but they are not scheme specific. Being a superclass, a simulation engine is able to collect parts of a state from the underlying components automatically using aliases and wildcards.

An important feature of the CoCoViLa platform is the ability to generate parts of the simulation program automatically, even at runtime, if needed. This adds flexibility in developing and reusing simulation models, as, for example, components can be synthesized and/or initialized lazily including fragments of event handlers.

4.4.2 Visualization

A visualization package in CoCoViLa consists of several components for plotting diagrams, two- and three- dimensional charts and also components for data input (sliders, etc.). Users can take individual components, import them into their own simulation packages and customize according to their specific needs.

4.4.3 Expert system

The expert system in CoCoViLa is developed as a standalone tool which may contain a number of decision tables. A decision table is implemented as a set of production rules. Several decision tables can be used simultaneously in one scheme. The expert system has the following features:

- XML format for storing tables in files;
- forward chaining inference engine for acquiring data from tables;
- graphical user interface for creating and managing tables, see Fig. 3;
- API to support querying tables from specifications of components.

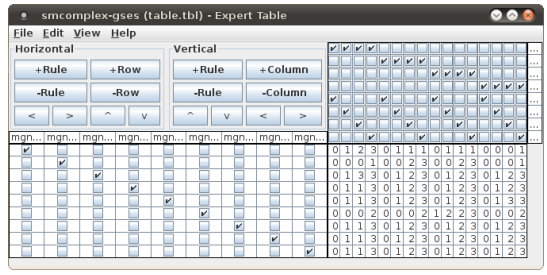


Figure 3: An example decision table opened in the expert system management GUI

5. APPLICATIONS

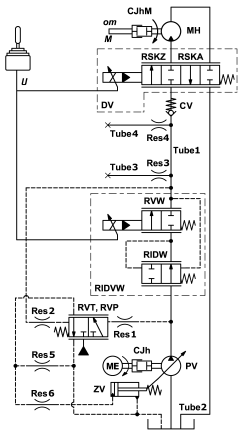
5.1 Simulation of hydraulic systems

The most sizable application of the CoCoViLa simulation tool is a modeling and simulation system for fluid power devices [5]. Fluid power systems assume a lot of drive and control tasks in machinery because of their high power density, flexible system character and required reliability. Computer modeling and simulation is an important phase in the design of such systems.

Multi-pole mathematical models and signal-flow graphs of hydraulic elements are used. This enables methodical, graphical representation of large and complicated chain systems. Simulated systems are decomposed into subsystems and functional elements. Multi-pole models of them may use programs, depending on the observed process (steady-state condition, frequency characteristics, transient responses).

Calculations are performed using multi-level method. In this way we can decompose large differential equation systems into smaller ones. First, calculations on the level of elements or subsystems are performed, thereafter variables between elements and subsystems are made congruent by iteration methods.

A library has been composed containing over hundred multi-pole models of fluid power elements that can be used for composing different schemes of fluid power devices and performing simulations. In particular modeling and simulation systems for hydraulic elements, electro-hydraulic servo systems and load sensing hydraulic systems have been developed. This is a joint work of Institute of Machinery and Institute of Cybernetics at Tallinn University of Technology.



PV – Pump; CJh – Clutch; M – Electric motor; ZV – Positioning cylinder; VP – Displacement of the pump control valve; RVP – Meter-in throttle edge of the control valve; RVT – Meter-out throttle edge of the control valve; Res1,. . . ,Res6 - Hydraulic resistors; RIDVW – Measuring valve with pressure compensator; RSKZ – Check valve with meter-in throttle edge of hydraulic motor; MH – Hydraulic motor; RSKA – Meter-out throttle edge of hydraulic motor; CJhM – Clutch with drive mechanism; Tube1,. . . ,Tube4 – Tubes.

Figure 4: Functional scheme of the hydraulic-mechanical load-sensing system

The example below considers simulation of a hydraulic load sensing system shown in Figure 4. Hierarchically built model of the device includes over 4500 dependencies represented by equations and Java methods. Typically, two kinds of simulations are performed: calculating steady state conditions and dynamic responses.

To give an impression of the visual features of CoCoViLa we present in Figure 5 a screenshot of a top level description of a hydraulic simulation problem of steady state conditions together with visualization of results of simulation shown in the lower right corner of the figure.

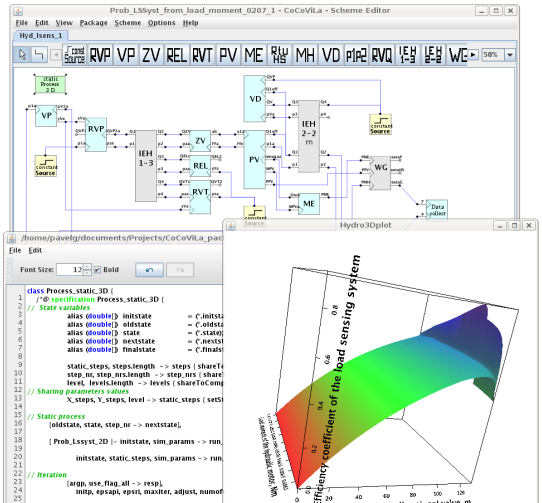


Figure 5: Simulation of a hydraulic-mechanical load-sensing system

The lower left window shows a piece of a text in the knowledge representation language. The text is obtained automatically from the visual description. It includes declarations of essential objects as well as logical rules for computing new values from given ones (lines from 10 to 24) and some explicitly given values (lines from 26 to 29) that have been entered through pop-up windows of the visual components. A toolbar with buttons RVP, VP etc. is scrollable and represents the concepts of the domain specific language for simulation of hydraulic systems.

Different simulation engines are used for calculating steady state conditions, 3D simulations and dynamic transient responses.

A special technique is used for calculating variables in loop dependences that can appear when a scheme of hydraulic device is composed from visual components. Splitting, using initial approximate values and iterative re-computing of the variables is used. Re-computing algorithms are constructed by the CoCoViLa program synthesizer as a result of solving corresponding subtasks. This avoids solving large equation systems during simulations.

The typical simulating task for calculating transient responses of the load-sensing system considered above contains:

- 37 classes, including 26 functional element classes;
- 16 variables that have to be iterated during the computations.

The automatically synthesized Java code for solving the simulation task for calculating dynamic transient responses that mainly consists of calls of methods has 4449 lines and includes 4 algorithms for solving different subtasks. Its synthesis takes less than a second on a typical 2 GHz laptop. Application described above is aimed at giving to the end user a convenient tool for experimenting with different model configurations and varying parameters of the models.

5.2 Simulations in cyber security

CoCoViLa has been applied in the cyber security domain for modeling and simulation of cyber attacks and selection of optimal countermeasures. Graph-based Automated Denial-of-Service Attack Response (GrADAR) [7] is an approach where the selection of attack responses is made according to an estimation of an impact of simulated counter-attack measures. CoCoViLa was used to create a GrADAR package for visual modeling and simulation of computer networks on the basis of information such as dependencies between system resources and their availability and workload values. The optimizer component of the package implements algorithms for automatic selection and application of response measures. This is a joint work of Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE (Germany), Cooperative Cyber Defence Centre of Excellence and Institute of Cybernetics (Tallinn, Estonia) [9].

Figure 6 shows a scheme in CoCoViLa representing a dependency graph of network resources with connections for propagating workload and availability values. The goal is to simulate and analyze the effect of response measures. CoCoViLa allows not only to enter the parameter values of components using the graphical user interface, but thanks to absence of restrictions on the usage of Java language and libraries, it also allows integration with other software, e.g., back-end management systems to receive live values into the running program synthesized from the scheme.

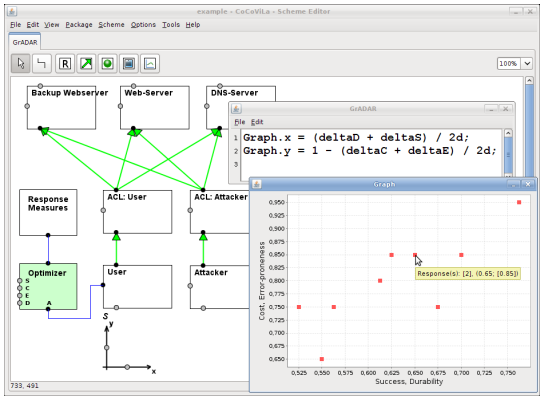


Figure 6: Visual specification of a response analysis problem in GrADAR package

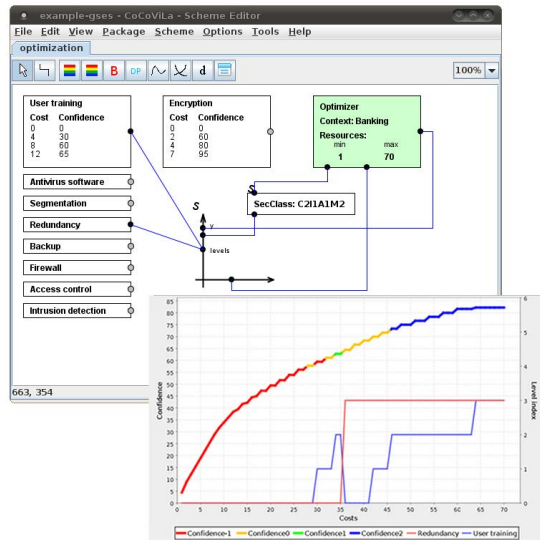


Figure 7: Visual specification and the result of simulation

5.3 Simulating graded security

This section concerns a work [8] in the field of modeling and simulation of graded security measures used in a banking security design. The graded security model is intended to help to determine reasonable or optimal sets of security measures according to the given security requirements. Currently, there are four main security goals in the model: confidentiality (C), integrity (I), availability (A) and satisfying mission criticality (M). For each goal, a certain level (0-3) is attached to specify the security requirements defined by a particular security class. As a simplified example, to achieve the security goals, the following groups of security measures might be selected for consideration: user training, antivirus software, segmentation, redundancy, backup, firewall, access control, intrusion detection, and encryption. The relative importance of a measures group can be specified by giving a weight value. In realistic scenarios studied so far the number of measures groups has been between 30 and 40.

Figure 7 shows a screenshot of the graded security expert system which consists of a visual specification language, implemented optimization algorithms and knowledge modules in the form of decision tables. In this package, security situations are described using the visual language. Suitable optimizers are applied to simulate all possible outcomes to find the Pareto-optimal set. During the simulation the optimizer queries decision tables for specific values. In the lower right corner of the figure the result of simulation, in the form of a Pareto curve and resource distribution curves, is shown.

As optimization procedures are contained in regular rich components, the algorithm used for a computation is easy to change by just replacing a component on the scheme. It can be useful to have a package contain several optimizers. Then the one making the most appropriate trade-offs for a particular experiment can be chosen by the user. In case of this package we have implemented two different optimizers, one employing brute-force and the other discrete dynamic programming algorithms. Adding a component implementing a genetic optimizer for the cases that cannot be handled by existing optimizers is anticipated in the future.

6. RELATED WORK

We are aware of a number of software products applicable for modeling and simulation. The ones we are mostly interested can be roughly divided into three categories: general-purpose modeling and simulation software, model-based application development software and large-scale discrete event simulation software. The common feature of most of these tools is visual specification capability by drawing schemes from components and connecting them to each other.

General purpose simulation products (Simulink [2], Scicos [1], Ptolemy II [10], etc.) typically possess built-in hybrid simulation engines and the simulation is flow-based — i.e., all the connecting arcs are directed and all the ports are either inputs or outputs. The models can be composed hierarchically and the components can be easily reused. Simulations are typically carried out on a virtual machine but also model translators exist to get code in some programming language (e.g., C) and to improve efficiency running the compiled code. Standard components (blocks) are grouped into packages (palettes). There exist large variety of blocks but also new blocks can be developed.

Model-based application develpment tools (e.g., MetaEdit [17]) have commonly a rich set of tools for program specification, analysis and verification. Taking the model-based approach to simulation development, it is easy to compose modeling and simulation applications with these tools. However, it typically needs more effort than using a dedicated tool.

Large-scale discrete event simulation tools (OPNET Modeler [14], OMNet++ [20], NS3 [15], etc.) are scalable simulators capable of handling large number of nodes and events. Their main concern is simulation performance and they are mostly used for in-detail analysis of network behavior and embedded systems, simulating all actions down to hardware level.

The CoCoViLa's planner provides flow based analysis similar to Simulink or Scicos, but its capability of constructing higher-order data flows allows us the development of model- and simulation-specific simulation engines in contrast to the mentioned tools. As the resulting simulation is always compiled into a single program it allows to achieve good simulation performance.

In comparison to other tools, CoCoViLa's support for handling of equations is very useful as this allows better reuse of rich components. For example, the ports are not defined strictly as inputs or outputs like it is the case with the tools belonging to the first two categories. It is up to the planner to decide which way the data flows in order to match the needs of the given problem—like it is the case also in real life—when defining a pipe it is not known beforehand in which direction the liquid flows in it.

Thanks to the hybrid simulation engine included in the toolbox, CoCoViLa's functionality is comparable to the general purpose simulation tools belonging to our first category. The hybrid simulation engine also covers the basic functionality of the tools belonging to the third category. However, it probably cannot always compete with specialized tools performance-wise.

Thus, although by its nature CoCoViLa is closest to the tools belonging to the second category, it is multifunctional and is more broadly applicable.

7. CONCLUSION

In this paper we described a flexible Java-based simulation platform CoCoViLa. It allows implementing simulation engines and other domain-specific simulation concepts as reusable components. The platform supports visual and model-based software development and uses structural synthesis of programs for translating declarative specifications of simulation problems into efficient executable code. To demonstrate the feasibility of our approach, several real-world applications were presented. Based on our experience, we suggest that CoCoViLa is suitable for creating and performing simulations in various engineering domains.

8. ACKNOWLEDGMENTS

This work was partially supported by the Estonian Ministry of Defence (grant No. 372/0807) and the target-financed theme No. 0322709s06 of the Estonian Ministry of Education and Research. The second and the fourth authors would like to thank the Estonian Information Technology Foundation's Tiger University+ program and the Estonian Doctoral School in ICT for supporting this work.

9. REFERENCES

[1] S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer, 2010.

[2] J. B. Dabney and T. L. Harman. *Mastering SIMULINK*. Prentice Hall, 2001.

[3] P. Grigorenko. *Higher-Order Attribute Semantics of Flat Languages*. PhD thesis, Tallinn University of Technology, 2010.

[4] P. Grigorenko, A. Saabas, and E. Tyugu. Cocovila - compiler-compiler for visual languages. *Electr. Notes Theor. Comput. Sci.*, 141(4):137–142, 2005.

[5] G. Grossschmidt and M. Harf. COCO-SIM – object-oriented multi-pole modelling and simulation environment for fluid power systems. part 2: Modelling and simulation of hydraulic-mechanical load-sensing system. *International Journal of Fluid Power*, 10(3):71–85, 2009.

[6] ILTP Library homepage. http://www.cs.uni-potsdam.de/ti/iltp.

[7] M. Jahnke, G. Klein, J. Tölle, and P. Martini. Protecting military networks with gradar - an approach for graph-based automated denial-of-service attack response. In *Proceedings of the International Military Communication Conference (MCC 2009)*, Prague, Czech Republic, Sept. 2009.

[8] J. Kivimaa, A. Ojamaa, and E. Tyugu. Graded security expert system. In *CRITIS*, pages 279–286, 2008.

[9] G. Klein, A. Ojamaa, P. Grigorenko, M. Jahnke, and E. Tyugu. Enhancing response selection in impact estimation approaches. In M. Amanowicz, editor, *Concepts and Implementations for Innovative Military Communications and Information Technologies*. Military University of Technology, Warsaw, 2010.

[10] X. Liu, Y. Xiong, and E. A. Lee. The ptolemy ii framework for visual languages. In *HCC*, 2001.

[11] R. Maigre, P. Küngas, M. Matskin, and E. Tyugu. Handling large web services models in a federated governmental information system. In *ICIW*, pages 626–631, 2008.

[12] G. Mints and E. Tyugu. The programming system PRIZ. In *Baltic Computer Science*, pages 1–17, 1991.

[13] A. Ojamaa and E. Tyugu. Rich components of extendable simulation platform. In H. R. Arabnia, editor, *MSV*, pages 121–127. CSREA Press, 2007.

[14] OPNET Technologies, Inc. OPNET Solutions. http://www.opnet.com/.

[15] S. Papanastasiou, J. Mittag, E. G. Ström, and H. Hartenstein. Bridging the gap between physical layer emulation and network simulation. In *Proceedings of IEEE WCNC Conference, 2010*, April 2010.

[16] A. Repenning, A. Ioannidou, and J. Zola. Agentsheets: End-user programmable simulations. *J. Artificial Societies and Social Simulation*, 3(3), 2000.

[17] J.-P. Tolvanen and S. Kelly. Metaedit+: defining and using integrated domain-specific modeling languages. In S. Arora and G. T. Leavens, editors, *OOPSLA Companion*, pages 819–820. ACM, 2009.

[18] E. Tyugu. The structural synthesis of programs. *Algorithms in Modern Mathematics and Computer Science*, pages 290–303, 1979.

[19] E. Tyugu, M. Matskin, and J. Penjam. Applications of structural synthesis of programs. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I*, pages 551–569, London, UK, 1999. Springer-Verlag.

[20] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *SimuTools*, page 60, 2008.

Publication II

Klein, Gabriel; Ojamaa, Andres; Grigorenko, Pavel; Jahnke, Marko; Tyugu, Enn (2010).

Enhancing Response Selection in Impact Estimation Approaches

Concepts and Implementations for Innovative Military Communications and Information Technologies. Ed. Amanowicz, Marek. Warsaw: Military University of Technology, 277–286.

Enhancing Response Selection in Impact Estimation Approaches

Gabriel Klein[1], Andres Ojamaa[2], Pavel Grigorenko[2],
Marko Jahnke[1], Enn Tyugu[3]

[1] Fraunhofer Institute for Communication, Information
Processing and Ergonomics FKIE
Neuenahrer Str. 20, 53343 Wachtberg, Germany

[2] Institute of Cybernetics at Tallinn Technical University
Akadeemia tee 21, 12618 Tallinn, Estonia

[3] Cooperative Cyber Defence Centre of Excellence
Filtri tee 12, 10132 Tallinn, Estonia

Abstract: The number of attacks against computer systems is steadily increasing. Network administration personnel often have a wide variety of response measures against these attacks at their disposal. In previous work, a methodology was introduced for efficiently assessing the effects of countermeasures on network resources before their actual application and thus determining the most appropriate response. Building on this, we now propose a method of dynamically weighting the metrics used to evaluate the different responses. Instead of a fixed linear combination of metrics we introduce Pareto optimal combinations of the individual metrics and the combined cost measure. This allows a more flexible way of emphasizing the importance of individual metrics in different situations. The methodology was prototypically implemented in CoCoViLa, a powerful simulation engine for visually specified optimization problems.
Keywords: Denial-of-service attacks, automated response, response evaluation, response metrics, Pareto optimality

1. Introduction

Along with the rising number of computer systems connected to the Internet which are infected with malware, the danger of large-scale denial-of-service attacks occurring also increases. To maximize the speed and reliability of response measures against such attacks, it is desirable to select and apply response measures automatically. In GrADAR (Graph-based Automated Denial-of-Service Attack Response), the selection of responses is made according to an estimation of the measures' impact on the protected system. Here, the impact is estimated according to different criteria, so-called metrics. Currently, the overall cost of a response

measure is defined as a linear combination of the different metric values in which each metric has a different weighting reflecting its relative importance. A higher flexibility can be attained by performing a Pareto optimization of the individual metric values as well as their linear combination. With this, multiple objectives regarding the metrics can be achieved; for example, a response measure with a maximal value for a certain metric can be chosen which also has a high overall rating.

The rest of this paper is structured as follows: Section 2 introduces related work in which Pareto optimality is used for multi-objective optimization. Thereafter, Section 3 gives a brief introduction to GrADAR. This is followed by a description of how Pareto optimization can be used to select response measures more flexibly and with regard to multiple objectives (Section 4). Section 5 gives details on our implementation in CoCoViLa, a visual simulation system. This is followed by Section 6 which presents first results of our work. Section 7 then summarizes our work and provides an outlook on further activities.

2. Related Work

In [1], Horn et al. introduce a Pareto criterion into the selection operator of a genetic algorithm to enable multi-objective optimization. As opposed to a scalar fitness function where the solution can be very sensitive to parameter changes, this allows a more robust selection of non-dominated solutions.

Douligeris [2] studies Pareto optimality in a telecommunications context. Here, flow control is managed by solving a problem with the two objectives maximal throughput and minimal delay. Pareto optimal solutions are then compared according to fairness.

From a network security point of view, the following contributions are interesting. Gu et al. [3] propose an intrusion detection system in which two different feature extraction approaches are used to construct event classifiers. The combination of the advantages of both systems into a single objective would require advance knowledge. Therefore, a multi-objective optimization is performed which yields Pareto optimal solutions.

In [4] and [5], Ojamaa et al. describe a graded security expert system which enables choosing security measures for information assurance. In the security model, the combination of the two objectives low cost and high confidence is achieved by an optimization technique based on dynamic programming. The user is presented with a Pareto optimality trade-off curve permitting the choice of the most appropriate security measures.

3. Graph-based Automated Denial-of-Service Attack Response

Graph-based Automated Denial-of-Service Attack Response (GrADAR, [6], [7]) is a framework for assessing the effect of response measures against denial-of-service

attacks on the availability of network services. This section describes the GrADAR model and introduces the required terminology.

In GrADAR, the so-called dependency graph $\hat{G} = (\hat{V}, \hat{E})$ models the ideal state of a network. Its vertices $\hat{V}$ correspond to the network resources and the edges $\hat{E}$ signify availability dependency relationships between the resources. Vertices $r_i \in \hat{V}$ are labeled with a dependency function D_{r_i} according to which, a resource's availability can be estimated based on the availability of antecedent resources. Additionally, the edges $e_{i,j} \in \hat{E}$ are labeled with a dependency weighting function $w_{i,j} : [0,1] \rightarrow [0,1]$ which signifies the degree to which resource r_i is dependent on resource r_j $(r_i \triangleright r_j)$.

A second graph $G = (V, E)$, the so-called accessibility graph, reflects the actual current state of the network. Its vertices $r_i \in V$ correspond to those in the dependency graph but are labeled with an availability value $A(r_i) \in [0,1]$. The set of edges E is a subset of $\hat{E}$ $(E \subseteq \hat{E})$ and an edge $e_{i,j} \in E$ reflects the ability of resource r_i to access another resource r_j.

Using information in both these graph structures now allows the estimation of availability values of resources for which availability is not directly observable. For a resource r_i, $r_i \triangleright r_j$ and $r_i \triangleright r_k$, the availability of r_i can be predicted using the following formula:

$$A(r_i) = D_{r_i}(w_{i,j}((A(r_j), w_{i,k}A(r_k))))$$

Figure 1 shows an example of both a dependency and an accessibility graph. Here, a user D (also modeled as a network resource) us dependent on the availability of a local operating system and a running HTTP service to perform some task, e. g. browsing a Web shop. The HTTP service itself is again dependent on the availability of the IP stack, in turn dependent on the operating system. The accessibility graph on the right shows a reduced availability of the IP resource, possibly due to an overloaded link to the nearest router (not displayed in the graph). Because of the availability dependency relationships between the resources, this results in a reduced availability of the user node, manifested, for example, by a reduction in speed of the user's browsing experience.

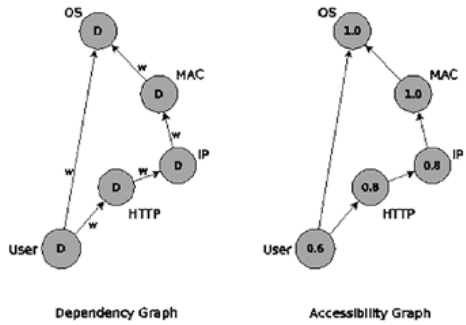


Figure 1. Example of a dependency graph and a corresponding accessibility graph

To estimate the effects of response measures on the availability of network services, each response is virtually applied to the model. The effects are then quantified and the most appropriate response is applied to the real-world network.

Virtual application of response measures is performed by modifying the dependency and accessibility graph in one or more of the following ways:

1. change availability values of nodes (G only),
2. adding/removing vertices ($\hat{G}$ and G), or
3. adding/removing edges ($\hat{G}$ and G).

After these changes have been made, the availability dependency relationships in the dependency graph need to be used by an update algorithm (e. g. a recursive depth-first search) to estimate the effect these changes have on the availability of other resources. For a more detailed description of availability propagation, please refer to [8].

Two possible response measures for the scenario depicted in Figure 1 could be the following:

1. Dynamic reallocation of the available bandwidth on the IP link. This could result in an increased availability of the local IP stack (corresponding to item 1 above).
2. Utilization of a second IP link to perform a form of load balancing. This would introduce a second IP node and a second MAC node into the graph, along with the corresponding availability dependency relationships (corresponding to items 2 and 3 above).

To select the most appropriate of the available response measures, they are evaluated with respect to multiple criteria, or metrics. There are currently four different metrics: success (δ_S), durability (δ_D), application costs (δ_C) and error-proneness (δ_E). These can be given individual weights by factors $w_S, w_D, w_C, w_E \in \mathbb{R}^+$.

Let $\Theta = \{\theta_1, \ldots, \theta_n\}$ be the set of all possible response measures. The best response measure θ_{best} is then determined by a suitability function which minimizes the costs and the error-proneness and maximizes the success and durability, i. e.

$$\theta_{best} = \arg\min_{\theta \in \Theta} \left(S(\theta) \right),$$

where $S(\theta) = w_C \cdot \delta_C(\theta) + w_E \cdot \delta_E(\theta) - w_S \cdot \delta_S(\theta) - w_D \cdot \delta_D(\theta)$ is the linear combination of metric values and $\delta_C, \delta_E, \delta_S, \delta_D$ are functions $\delta_i : \Theta \to \mathbb{R}$ which represent the metrics.

Figure 2 shows an overview of the GrADAR approach. The ideal state of the network captured in the dependency graph is augmented with availability information for some resources provided by an implemented intrusion detection or network management system. An update algorithm utilizing the availability dependency relationships between resources is used to estimate availability values for resources for which no values were provided by the IDS/NMS.

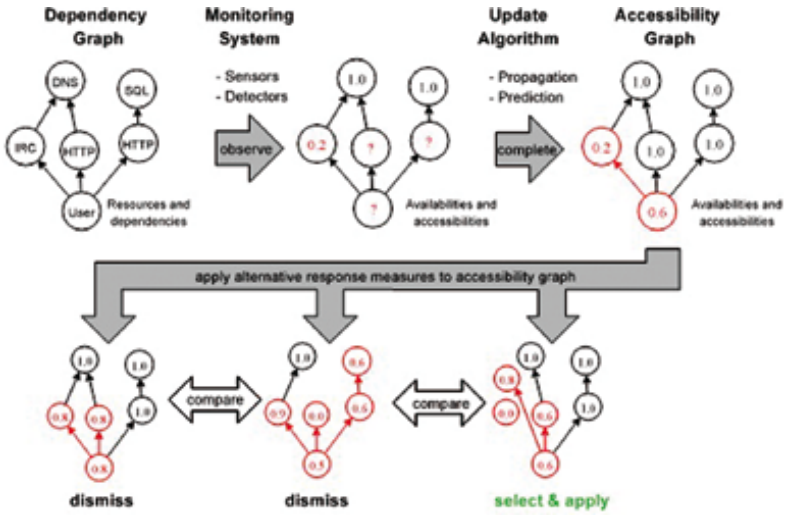


Figure 2. Overview of the GrADAR approach

Real-world response measures correspond to transformations of the two graphs. For each possible response measure, the graphs are individually modified. The resulting response graphs provide a measure for the resulting availability after the application of the corresponding countermeasure. However, this is only one of the possible metrics according to which countermeasures can be evaluated (see above). After an appropriate response measure has been chosen and applied, the corresponding response graph serves as dependency graph for the next iteration of GrADAR.

4. Pareto Optimal Response Selection

Pareto optimality or efficiency is a concept originating in economics. Broadly speaking, a Pareto optimum of a group of individuals is a state in which any change to the benefit of an individual would at the same time be to the detriment of another individual. More formally, an n-dimensional tuple $x_1,\ldots,x_n \in A$ is a Pareto optimum of the set A if there is no tuple $y_1,\ldots,y_n \in A$ with $y_i \geq x_i$ for $i=1,\ldots,n$ where "≥" is a superiority relation. The set of all Pareto optimal outcomes is called a Pareto set.

To extend the rather static assessment of response measures through the weighted linear combination of individual metrics, we propose to also analyse responses by presenting the results in the form of Pareto sets. This is theoretically possible for all available metrics, i. e. for all response measures θ_i, the tuple $\left(\delta_C(\theta_i),\delta_E(\theta_i),\delta_S(\theta_i),\delta_D(\theta_i),S(\theta_i)\right)$ can be represented. Note that subsequently, the weighted combination of metrics $S(\theta)$ will be treated as a metric as well, signifying the overall "cost" of the response measure. However, to retain overall manageability, analysis should be restricted to two or three metrics. We can, for example, plot the best possible values of $S(\theta)$ for certain response vectors against the values

of selected metrics, e. g. application costs, for these responses. A reasonable choice would be to compose the metrics reflecting gains, $S^+(\theta) = \delta_S(\theta) + \delta_D(\theta)$, and those reflecting the losses, $S^-(\theta) = \delta_C(\theta) + \delta_E(\theta)$, and to plot the curve relating $S^+(\theta)$ and $S^-(\theta)$. The final response choice will then explicitly take into account both the overall quality and the costs.

It is important to have a convenient way of selecting different Pareto variables and plotting different Pareto curves. The next section gives a brief overview of a tool developed for this purpose.

5. Model-based Implementation

The aim of the present approach is to develop an automatic response selection method by experimenting with different ways of the response selection. To facilitate the experiments, we have developed a visual model-based software tool for representing accessibility graphs and problems on these graphs. This is a GrADAR software package developed for the CoCoViLa platform [9]. The package provides assets for specifying response selection problems and for high-level control of computations on the graph. It contains components for resources, optimization methods and for visualization of results. CoCoViLa supports problem solving on higher-order constraint networks [10] that can be easily used for propagating availability values on the accessibility graphs.

The first application of the software was to analyze the effect of response measures by automatically propagating workload values (red arrows) and availability values (green arrows) of resources. This is shown in Figure 3. The problem was visually specified as a scheme that was a union of dependency and accessibility graphs extended with an analysis component (Propagator). Nodes representing resources had a number of parameters that were observed and adjusted in a property window of a node.

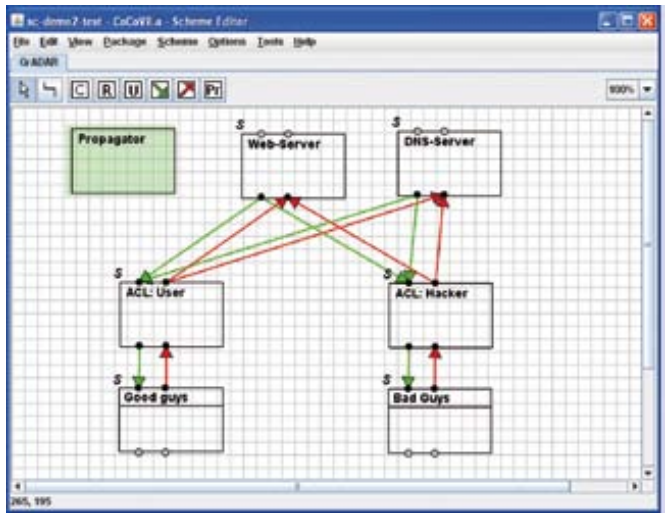


Figure 3. Visual specification of a response analysis problem

There are two possibilities for specifying resource availability values. On the one hand, values can be manually entered in the resource properties window. This supports offline simulations of the effects of countermeasures on static resource scenarios and can be useful for trial-and-error determination of novel countermeasures. On the other hand, there are also interfaces to arbitrary back-end management systems from which live values can be obtained. Thus, the simulation engine has a dual use as a monitoring system operating on real-life values. This can aid in real-time countermeasure evaluation for network administration personnel.

Figure 3 shows a visual specification of the response analysis problem. Its menu bar contains buttons for all types of components and connectors. A specification was built by using buttons of the menu bar, and by introducing parameter values of components through their pop-up property windows. A properties window for a resource node (Web-Server) is shown on the right side in Figure 4.

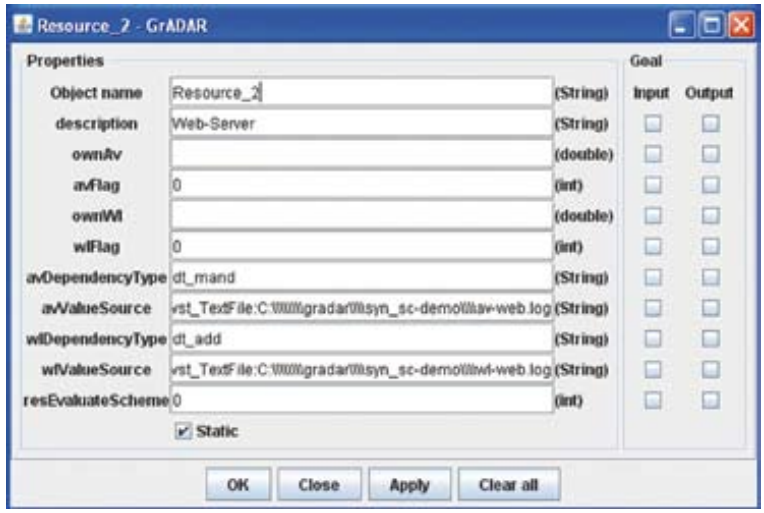


Figure 4. Properties window of the Web server resource

The problem can be specified in a textual language as well. In fact, a textual specification is always automatically generated from a visual specification, if the latter has been given. The ability to generate the scheme in textual format is especially useful because scheme generation can be automated and processed offline.

6. First Results

The main application of the developed software is calculation of various Pareto sets, using an optimization component for selecting the best solution from a given set of possible response measures. This problem is described by accessibility graph, extended with visualization, optimization and response measures nodes; see Figure 5 for a simplified example.

The developed GrADAR package enables decision-making (choosing a response) by first determining a set of admissible responses and thereafter either finding the best response or plotting a Pareto curve to help with the choice. The computations are performed as follows. The set of possible responses Θ is a set of tuples constructed from possible response measures for the resource nodes of the accessibility graph. A tuple of response measures constitute the description of a response θ, we call it a response vector. The optimizing component is able to produce all required values of the response vector and distribute its components to the resource nodes for calculations. The results of calculations are collected from the resource nodes and passed back to the optimizing component. This collection and distribution of responses is described in the optimizing component simply by the following CoCoViLa statement:

```
alias responseVector = (*.response);
```

where `response` must be the name of a response action in each resource node.

At the present stage we use a brute-force search for determining the best response for given arguments, generating all possible values of response θ. A Pareto set of pairs (x, y) is constructed as follows. Values x and y of the Pareto coordinates are calculated for all possible values of θ. The response θ' with the best value of y is selected for each given value of x. A Pareto set (the set of selected points (x, y)) is plotted. Also, a table can be constructed with rows representing a response θ' for each point (x, y). As we have noted in Section 3, different metrics can be used as the variables x and y. Figure 5 shows a visual specification of a problem and a Pareto set for x representing normalized gains S^+ and $y = 1 - S^-$, where S^- represents normalized costs. Another potentially interesting Pareto set is for x representing δ_S (success of the countermeasure) and y representing S, the overall cost measure for the countermeasure. This would enable an administrator to choose a response measure which maximizes the resulting network availability while minimizing the overall cost.

We would like to emphasize that the user can quickly analyze multiple trade-off situations by connecting various ports of the optimizer component outputting different metric values to the ports of the graph component. New, arbitrarily complex metrics can be defined using equations and existing Java methods in the specification window.

In order to be able to analyze responses that introduce new elements into the graph, we use a supergraph of the accessibility graph that includes all possible extensions, e. g. the servers that can be added as responses. When the availability of all these resources is zero, we get the initial accessibility graph that can be extended. Actually, the node Backup Webserver in Figure 5 is just a node with zero initial availability, i. e. it can be added to the network as a response.

The graph window in Figure 5 shows the points of the calculated Pareto set as red rectangles (only the points with the highest y value for each x value are visible). Each point represents the estimated outcome of applying a response. The graph

displays a tool tip on each rectangle which contains the index and the exact x and y values of the corresponding response (the tool tip also shows the values for points with the same x value not in the Pareto set). The index can be used for looking up the response steps leading to this outcome.

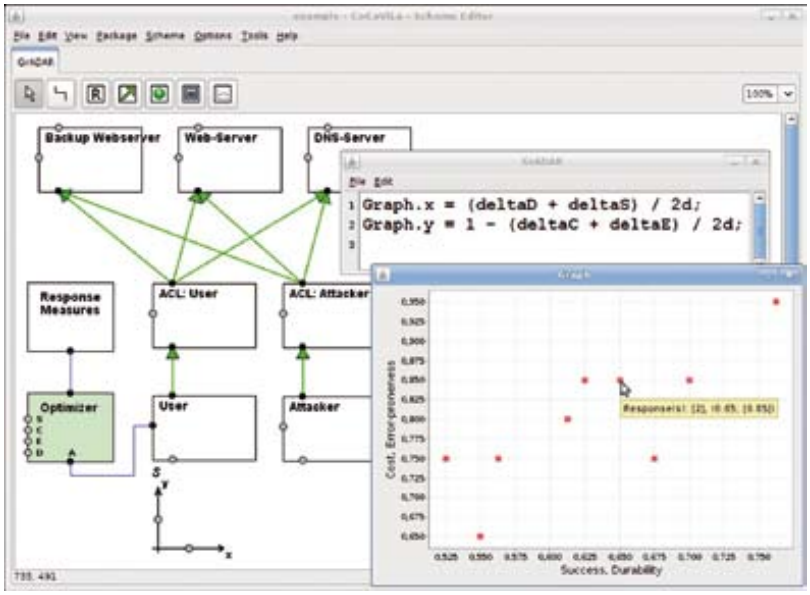


Figure 5. Plot of the Pareto set (gains, costs)

7. Conclusion and Further Work

We have presented a methodology and prototypical implementation for dynamically modifying the weighting of metrics for evaluating the effects of pre-defined countermeasures against computer network attacks. Instead of a fixed combination of metric values, Pareto sets now allow more flexible and more differentiated determination of the most appropriate reaction to a detected attack.

Currently, only the evaluation of pre-defined countermeasures is supported. However, it is possible that through recombination of elementary response steps, new, more sophisticated responses can be generated. This includes the definition of a permissible order in which response steps can be concatenated and the elimination of infeasible or erroneous results. This requires further research into the necessary changes to the model.

Acknowledgements

The authors would like to thank the Federal Office for Information Management and Information Technology of the German Armed Forces, the Cooperative Cyber Defence Centre of Excellence, the Estonian Defence Forces Training and

Development Centre of Communication and Information Systems, and the Estonian Ministry of Defence (grant No. 372/0807) for the support of this work. The second author would like to thank the Estonian Information Technology Foundation and the Tiger University programme for partial support of this work.

REFERENCES

[1] Horn J., Nafpliotis N., and Goldberg D., A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pages 82-87, 1994.

[2] Douligeris C., Multiobjective flow control in telecommunication networks. In: INFOCOM '92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE, volume 1, pages 303-312, May 1992.

[3] Gu Y., Zhou B., and Zhao J., PCA-ICA ensembled intrusion detection system by pareto-optimal optimization. Inform. Technol. J., 7:510-515, 2008.

[4] Ojamaa A., Tyugu E., and Kivimaa J., Pareto-optimal situation analysis for selection of security measures. In: Proceedings of the 4th IEEE Workshop on Situation Management SIMA 2008, San Diego, CA, USA, November 2008.

[5] Kivimaa J., Ojamaa A., and Tyugu E., Graded security expert system. In: R. Setola and S. Geretshuber, editors, Critical Information Infrastructures Security, volume 5508 of LNCS, pages 279-286. Springer-Verlag, 2009.

[6] Jahnke M., Tölle J., Thul C., and Martini P., Validating GrADAR – An Approach for Graph-based Automated DoS Attack Response. In: Proceedings of the 34th IEEE Conference on Local Computer Networks (LCN2009), Zurich, Switzerland, 2009.

[7] Jahnke M., Klein G., Tölle J., and Martini P., Protecting Military Networks with GrADAR – Graph-based Automated DoS Attack Response. In: Proceedings of the Military Communication Conference 2009, Prague, Czech Republic, 2009.

[8] Jahnke M., Graph-based Automated Denial-of-Service Attack Response. PhD thesis, University of Bonn, 2009.

[9] Grigorenko P., Saabas A., and Tyugu E., Visual tool for generative programming. In: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering. ACM Press, 2005.

[10] Tyugu E. and Uustalu T., Higher-order functional constraint networks. In: Constraint Programming, volume 131 of NATO ASI Series F: Computer and System Sciences, pages 116-139. Springer-Verlag, 1994.

Publication III

Ojamaa, Andres (2009).

Hybrid Simulation of Large Networks

Proceedings of the 2009 International Conference on Modeling, Simulation & Visualization Methods, MSV 2009. Ed. Arabnia, Hamid R.; Deligiannidis, Leonidas. Las Vegas: CSREA Press, 219–225.

Hybrid Simulation of Large Networks

Andres Ojamaa
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia

Abstract— *In this paper a new method for hybrid simulation of large computer networks is introduced. This technique allows to combine two different existing simulation methods: discrete event simulation and time-stepped continuous process simulation. The blending of methods with distinct properties gives more options in choosing the appropriate trade-off level between performance and accuracy that is needed for scalability. The novelty is that the hybrid method uses automatic synthesis of programs for generating parts of the simulation program from high-level logical specifications. A prototype implementation of a simulation environment supporting the hybrid simulation method is also described. The implementation is built on a Java-based simulation tool CoCoViLa that supports visual specification of simulation problems and program synthesis.*

Keywords: Hybrid simulation, large-scale network simulation, software components, visual specification, Java-based modelers

1. Introduction

Simulation of large computer networks has remained a challenging task. This is the case despite many efforts that have been made to develop efficient tools and accurate methods. Although one can see some improvements in this field, the growth of problems still outpaces the development of the modeling and simulation methods and tools.

The prelevant approach to simulation of computer networks is discrete event simulation. Several well known discrete event network simulators exist. Network simulators that are popular in the academic community include, for example, NS2/NS3 [1] and the INET Framework. The latter is based on the OMNeT++ discrete event simulation environment [2]. From commercial offerings, besides others, there are OPNET Modeler and the QualNet product family.

Packet-level discrete event simulations can produce very accurate results provided that the used models are correct. However, this accuracy comes with a relatively high computational cost. While this is a workable solution for small scale experiments, it is not commonly suitable for large networks. The limiting factor of packet-based simulation of large networks is that network sizes and bandwidths are increasing rapidly but the average size of packets for a certain protocol and workload is quite stable. Therefore the number of packets traversing the simulated network also increases very quickly with the growth of the network model. The complexity of the simulation depends at least linearly on the number of packets flowing in the simulation. While packet-level discrete-event simulations can be executed in parallel, it is hard to parallelize the simulation effectively when many nodes are connected with high speed and low latency links. This is because small propagation delays result in time synchronization overhead becoming more and more prohibitive.

Advanced techniques have been proposed to speed up packet-level discrete-event simulations. The obvious way to achieve that is to find solutions for reducing the number of events that have to be processed. One solution from this class is based on the observation that a large amount of network traffic usually involves sequences of packets positioned closely in time that are exchanged between the same two endpoints. Therefore, instead of individual packets, simulations can treat sequences of packets or packet trains as the basic unit of network traffic to reduce processing workload. A packet train model was introduced in [12].

One of the key qualities of large network simulations is scalability. Better scalability is generally achieved by abstracting away the low level details (e.g., simulation events triggered by transmission of a single packet in a high bandwidth link) where it can be done without affecting fidelity of the results too much. The already mentioned packet train method is one step in this direction. However, considering the limits of packet-based discrete-event simulations a number of alternative approaches have been developed over the years. These methods include giving up the packet-level simulation and replacing it completely with analytical approximations of network behavior, as well as other simplifications. A common abstraction over packets is so-called continuous fluid model and variations of it (see, e.g., [14]).

The main focus of our work is on the integration of a hybrid simulation method with automatic synthesis of programs. This approach gives all the benefits of hybrid simulation. Adding automatic synthesis capabilities brings its own benefits, mainly flexibility, faster development of simulation experiments and efficiency of execution. Large simulation program can be synthesized from compact, modular and high-level specifications. Also programs with different emphasis (e.g., performance, accuracy) can be generated from the same high-level specification. This hybrid technique is rather general and applicable to different problem domains,

not restricted to just network simulations.

The details of modeling specific network protocols analytically, methods for accounting of interactions between packet and fluid flows, calculating end to end delays etc are however out of the scope of this paper. These questions have been covered in different papers, see, for example: [3], [15], [17].

The hybrid simulation method presented here is based on the assumption that most of the traffic volume is modeled as fluid and simulated as a continuous process. It is also assumed that the clocks of discrete event process and time-stepped continuous process are not synchronized during a time step, and this does not result in significant errors.

The rest of the paper is structured as follows. We will continue with a brief overview of related work and a discussion of various approaches to hybrid network simulation. After that the basics of the simulation environment CoCoViLa are introduced. In the following part, Section 4, the hybrid simulation method with automatic program synthesis is presented along with a simplified example of hybrid network simulation.

2. Related Work

A hybrid packet/fluid network simulation method and environment is presented in [19]. The authors have integrated slightly modified versions of two different simulation tools: *pdns* for simulating foreground packet-based flows, and HDCF-NS fluid simulator for simulating background traffic. While experiments show considerable speed-ups in some cases, the performance is not very good in cases where a large number of minor fluctuations in flow rates (the "ripple effect") was triggered in the fluid simulator.

In [4] a general hybrid systems modeling framework of describing the flow of traffic in communication networks is presented. To characterize network behavior, these models use averaging to continuously approximate discrete variables such as congestion window and queue size. The results are validated against results obtained with the NS2 simulator. Hybrid models of TCP and UDP are described along with the framework. A discussion on computational complexity is also presented where the differences between hybrid and pure packet-based simulations are outlined.

In [16] a design of a hybrid Java simulator, called HNS is outlined. HNS simulates the movement of workload in a queuing network, where transactions may be of two types: traditional discrete packet transactions and continuous fluid transactions. HNS introduces and utilizes a so-called streamlining methodology to avoid situations where the "ripple effect" causes slowdowns of the simulation. HNS considers only static network models and, as fluid flow rate changes are propagated without delays, loops in HNS fluid flows are not allowed. The proposed method shows good performance when there are few packet-based flows present in the simulation.

A hybrid packet/fluid model where packet and fluid flows coexist and interact is presented in [13]. The goal is to speed up network simulations where background traffic is modeled as fluid flows. Performance and accuracy is compared to packet-level discrete-event simulations.

Another time-stepped hybrid simulation algorithm and its implementation in a simulation environment NetScale is presented in [5]. The authors claim to be able to simulate up to one million competing flows and network elements. Results of example problems are compared to results published elsewhere and scalability and excellent accuracy compared to packet-level discrete-event simulations is reported.

PtolemyII [11] is a Java-based hybrid simulation environment. It has a graphical user interface and reusable components are used for composition of simulation programs. This framework features Higher Order Components (HOCs) which can programmatically rewrite the scheme: add and remove components, copy parts of the scheme, etc. However, in HOCs there is no automatic synthesis of programs involved as the implementation of HOCs is pre-programmed.

To the best of our knowledge currently no other method or tool uses hybrid simulation method with synthesis of simulation programs.

Our implementation of the hybrid method is based on a simulation environment CoCoViLa [6] that supports visual specification of simulation problems and automatic synthesis of programs. CoCoViLa has been proven to be an effective tool for continuous simulation of large hydraulic systems [8]. The systems simulated there are essentially complex dynamic systems which have their behavior defined using ordinary differential equations. The modeling and simulation approach is similar to fluid models of computer network simulations.

However, until recently the CoCoViLa system lacked the concept of an event and as such was not useful for discrete-event and hybrid simulations. A discrete-event simulation engine was implemented and integrated into CoCoViLa system to obtain the means for developing hybrid simulations.

3. Simulation Environment

Let us now describe the software environment that is used as a basis of the implementation of the hybrid simulation method. CoCoViLa is a model-based software development and simulation platform. It provides tools for developing reusable software components, constructing simulation packages, specifying simulation problems and running them as hybrid processes.

Simulation packages in CoCoViLa consist of software components called rich components [18]. Rich components are descriptions of domain-specific concepts that are used in simulation. In a nutshell, these components are composed of the following four parts. First, there is a visual part for interaction with a user. Second, the logical part containing a high-

level specification that enables automatic composition of a simulation program. The third part is a program component which defines Java methods that implement computations. Finally, a rich component may have a daemon that provides interactive properties allowing to develop flexible interfaces to simulation programs.

From a user's point of view the toolbox consists of two programs: *Class Editor* for the component/package development and *Scheme Editor* for visual specification of simulation problems and for performing the simulation.

3.1 Class Editor

Class Editor supports a language designer in defining visual aspects of concepts, and also their logical and interactive aspects. In our terms — Class Editor is a tool for developing visual parts of rich components and for binding them with other parts developed as Java classes. It provides dialog windows for defining properties of ports (connection points), defining component properties, e.g., the fields that will be shown in a dialog windows in the Scheme Editor.

3.2 Scheme Editor

Scheme Editor is a tool for performing simulations. It is intended for developing schemes of simulated systems, compiling a simulation program and running it. It is used for compiling programs from the schemes according to the specified semantics of a particular domain. It provides an interface for visual programming — putting together a scheme from visual images of concepts. After loading a package into the Scheme Editor workspace, we get an environment for a particular simulation domain that allows a user to draw, edit and compose schemes through language-specific menus and toolbars. It allows one to compile a simulation program from a given scheme and to control the simulation process through the daemons of rich components.

Fig. 1 shows the Scheme Editor in use, when a package for simulating simple hybrid queuing networks has been loaded. The scheme describes a network of three traffic generators and a bounded buffer. This scheme has been composed by connecting ports of components of the following types: *TrafGen*, *Buf*, *Clock* and *Graph*. The toolbar at the top of the scheme is for adding objects and connections to the scheme. The component *Proc* implements the hybrid simulation process which is discussed in detail in the following sections. A pop-up window and a pop-up menu are also visible in the figure. The pop-up window is for instantiating object attributes, the pop-up menu is for manipulating the scheme — deleting and arranging objects etc. The Scheme Editor is fully syntax directed and the correctness of the scheme is forced during editing.

Some words must be said about the compilation of a simulation program. First of all, a scheme is translated into a textual specification that represents the same information that the scheme does. The textual specification language has a precise semantics implemented by means of an attribute technique. An efficient method of attribute evaluation for specification languages [7] has been implemented in CoCoViLa. This is dynamic evaluation of attributes on higher-order attribute models that are used for describing the semantics of visual languages.

Fig. 2 shows a textual specification obtained from the scheme of the simple example shown in Fig. 1. In Fig. 2 in the window on the left, one can see that the specification is created as a logical part of a rich class *Hns*. On the right, the figure shows also a fragment of a simulation algorithm synthesized from the specification. This is algorithm is compiled into a Java class that in the present case is 144 lines of code and is not shown here. This Java class can be compiled and run in the CoCoViLa environment. The scheme and the algorithm windows shown in Fig. 2 are useful for debugging a specification, but they are not needed when solving simulation problems in CoCoViLa, because the user interacts with a simulation program through the Scheme Editor main window shown in Fig. 1.

3.3 Logical and Program Parts of a Rich Component

Logical part, also called metainterface, and program component constitute jointly a single Java class. The logical part is intended for expressing attribute semantics of the model described by means of a rich class or by a composition of rich classes. It is included in the Java class as a comment, another part of the class presents data and methods that can be used as specified in the logical part. The logical part is a specification written in a simple specification language that includes:

- Specifications of variables
  $type\ id, [id, ...]$
- Bindings
  $var1 = var2$
- Axioms
  $precondition \rightarrow postcondition\{implementation\}$

Types can be primitive Java types (`int`, `double` etc), Java classes or rich classes. *Bindings* are used for structural composition, to specify the equality of two variables $var1$ and $var2$ (these can be components of other variables declared in the specification).

Axioms are specifications of methods of the class. The *preconditions* of axioms can be conjunctions of propositional variables and implications of conjunctions of propositional variables. The *postconditions* are conjunctions of propositional variables. Name of any variable from specification can be used as a propositional variable, denoting that value of the variable can be computed. An implication in a precondition denotes a goal — a computational problem whose algorithm has to be synthesized before the method with the precondition can be applied. This logic has been tested in several practically applied synthesizers, in particular, in the

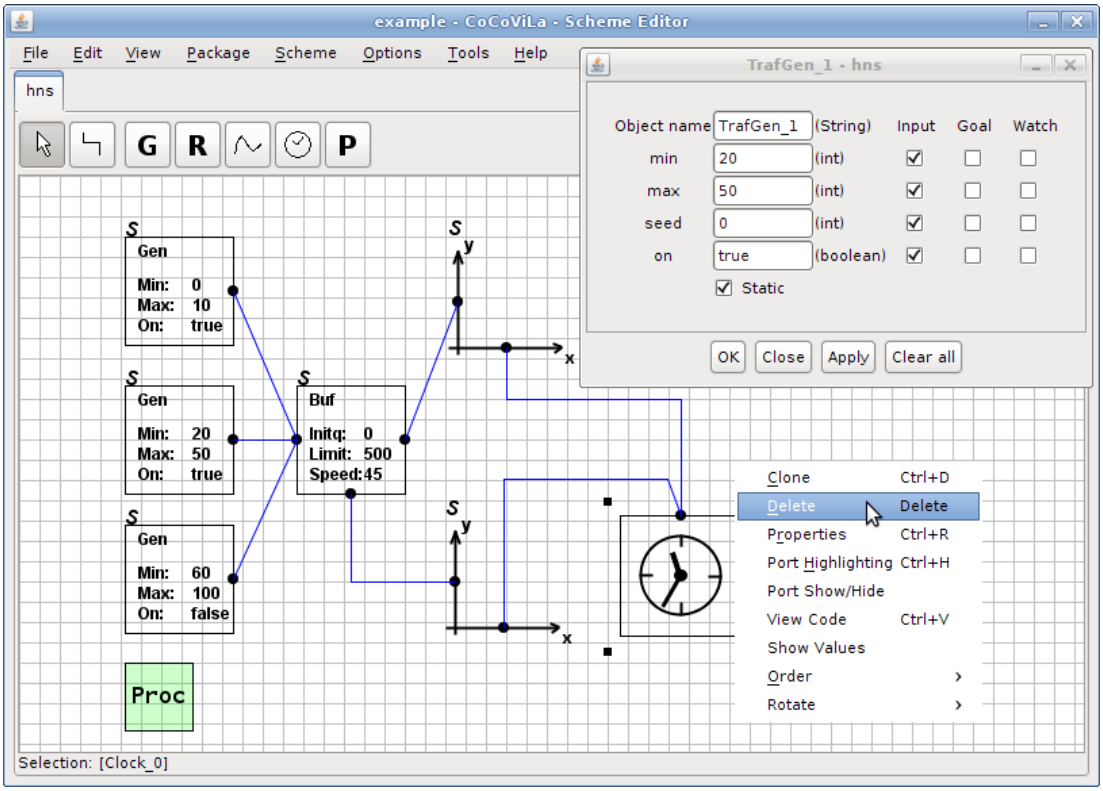


Fig. 1: Scheme Editor window

NUT system [20]. *Implementation* is a name of a method of the class being specified.

Aliases correspond to structural relations that binds tuples of attributes, i.e. $x = (x_1, \ldots, x_m)$. This feature can be used when more than one variable needs to be bound via a port. It is also useful for hierarchical composition, as aliases may contain other aliases, possibly defined using wildcards.

Wildcards are used for defining special aliases that dynamically bind variables having the same name that are defined in other metainterfaces of components declared on the same level with the wildcard. For example:

$$Router\ r1, r2;\ alias\ x = (*.queue).$$

For convenience of the user simple algebraic equations have been included in the specification language:

$$expression_1 = expression_2,$$

where expressions are arithmetic expressions of a restricted form that depends on the equation solver. An equation can be considered as a specification both of methods and of axioms. The methods represented by an equation are those that solve the equation with respect to its variables.

Metaclasses also support object oriented inheritance. Given a metainterface A, metainterface B inherits all the variables, bindings, and axioms defined in A using the $super$ keyword at the declaration of the metainterface: $B\ super\ A$. A $superclass$ is an instance of a rich component from which the scheme inherits the logical part and the program part.

A specification in this language describes a higher-order attribute model. Higher order means that inputs of attribute dependencies can be functions that must be synthesized on the attribute model. The attribute evaluation on higher-order attribute models is described in [7]. This method provides a dynamically composed attribute evaluator for a given attribute model and a given problem in the form "given $x_1, \ldots, x_m$ compute $y_1, \ldots, y_n$". This technique is efficient and reliable. In practice, composition of an attribute evaluator takes a fraction of second.

4. The Hybrid Method

Let us now describe the hybrid simulation method. This method integrates conventional discrete-event simulation and

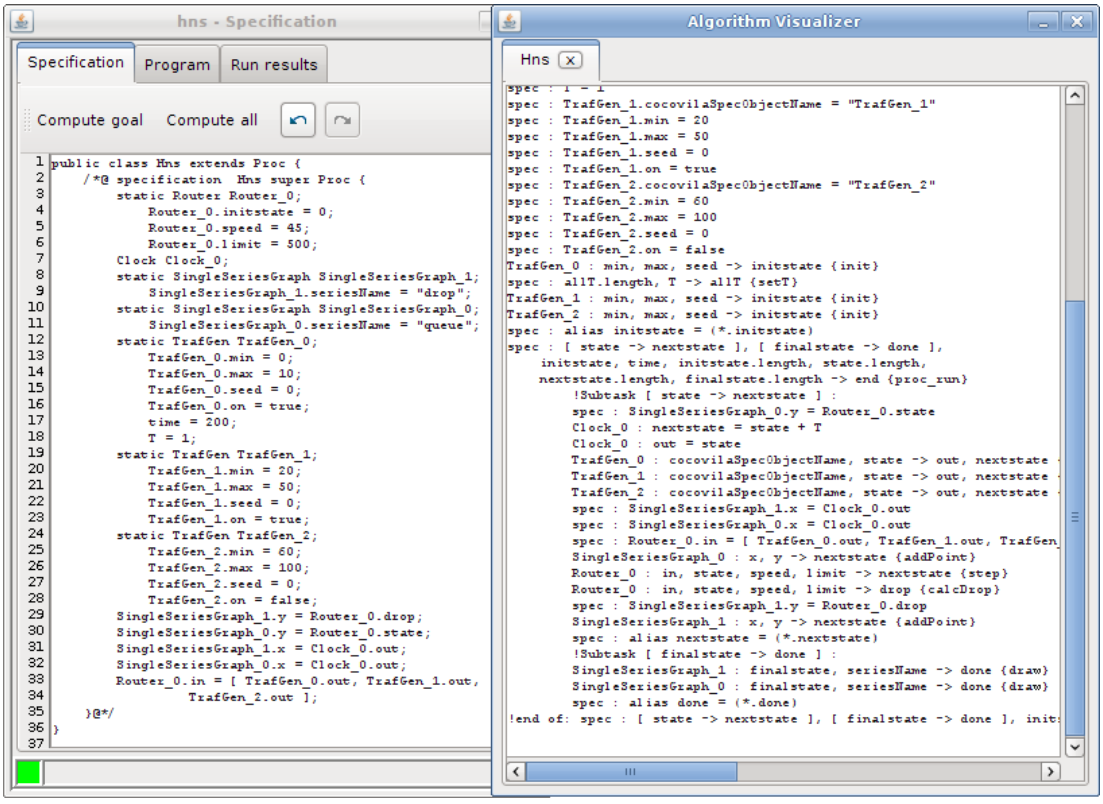


Fig. 2: Textual specification and synthesized algorithm

continuous time-stepped simulation methods in order to be able to simulate hybrid processes. As a novel feature, parts of the hybrid simulation program are synthesized automatically. The goal of developing this method is to make describing large systems easier and more flexible. First, it is shown how continuous and discrete processes are modeled in our framework, and how program synthesis is used. After that, integration of both modeling approaches is described.

4.1 Continuous Process

A continuous process is modeled as evolution of the system state s in time. The state consists of the attribute values of the system components. The process is computed in discrete time steps using a transition function f. The transition function is fixed, and it uses the state s_n corresponding to time step t_n as an input to compute the next system state s_{n+1} at the next time step t_{n+1}:

$$s_{n+1} = f(s_n) \ .$$

During computation of the transition function the simulation time advances by a time quantum T to the next time step value. The process is started at time step t_0 using a given initial system state s_0. The process stops at time step t_f after computing the final state s_f.

In our case, the transition function is synthesized from logical specification. The computation is specified as an axiom with an empty implementation in the following form: $state \rightarrow nextstate$, where $state$ and $nextstate$ are aliases. If a corresponding function can be derived on the given model, a Java class is generated that implements this computation. This class can then be used as an input for other methods where its *compute* method is invoked with specific argument values.

The synthesized program represented by a Java object is passed to a Java method that controls the execution of the continuous simulation process. Typically this execution loop is implemented in the superclass of a scheme. As an example, in a simple case the metainterface of the superclass contains the following specification:

```
  alias initstate  = (*.initstate);
  alias state      = (*.state);
  alias nextstate  = (*.nextstate);
  alias finalstate = (*.finalstate);
  long  t, T;
  [state -> nextstate], initstate, t, T
               -> finalstate {proc_run};
```

The method $proc\_run$ might be implemented as follows:

```
public Object[] proc_run(Subtask f,
    Object[] initstate, long t, long T) {

  Object[] state = new Object { initstate };
  try {
    for (long i = 0; i <= t; i += T) {
      state = f.run(state);
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
  return state;
}
```

In this example, four aliases are defined to bind state variables. All components that need to keep a state also define attributes with the same names. State variables defined in subcomponents are included in the top level state variables.

An implementation of the transition function specified as $state \rightarrow nextstate$ has to be synthesized before the method $proc\_run$ can be called. In addition, the values of $initstate$, t, and T must be given or computed by a generated algorithm. In the method $proc\_run$ the generated transition function is called for each time step, until the final state is reached and returned.

4.2 Discrete Process

In contrast to time-driven execution of continuous processes, the execution of discrete processes is event-driven. In case of discrete-event simulation there is a simulation time ordered event queue. Events are added to the queue and scheduled for processing at a specific point in simulation time. After the event with the earliest timestamp is processed by the simulation engine, the simulation time is advanced up to the timestamp of the next event at the queue. New events may get scheduled or already scheduled events may get removed from the queue as a result of processing an event.

It is important to note that while an event is processed the simulation time does not advance. In other words, the processing of an event is instantaneous with respect to the simulation time.

A discrete event simulation engine $Simulator$ was implemented in CoCoViLa. The simulator is a Java class that has the essential methods for implementing discrete processes. The methods include: $scheduleAt$, $getNextEventTime$, $step$, and run. Also, a base class for implementing events was added – $SimEvent$ – which has a two argument constructor. The arguments to the constructor are a reference to a $Runnable$ object and a timestamp. The simplest use case of the simulation engine is

```
Simulator.scheduleAt(1.0, new MyEvent());
Simulator.run();
```

where $MyEvent$ is a Java class that implements the $Runnable$ interface. The method run of this event class instance is executed at simulation time $t = 1.0$. The method $Simulator.run$ blocks until all events have been processed or the simulation is stopped explicitly by an event.

4.3 Hybrid Process

In order to integrate the two methods described above, a protocol for interleaving the processing of discrete events and continuous simulation steps and synchronizing the simulation time has to be defined. Based on our assumptions and goals the following approach was chosen. Before executing the continuous transition function at t_n to reach simulation time t_{n+1} and corresponding state, all events scheduled for the time interval $[t_n, t_{n+1})$ have to be processed. To achieve that, the simulation loop presented in Section 4.1 was amended by adding the following lines of code at the beginning of the for-loop:

```
while (Simulator.getNextEventTime() > -1
    && Simulator.getNextEventTime()
        < i + T) {
  Simulator.step();
}
```

Depending of the requirements of a particular simulation task a different solution to this problem might be appropriate. In our framework, it is easy to provide several implementations of the simulation process as rich components. The implementation can then be changed by just replacing the scheme superclass component. More about correctness issues of integrating fluid and packet models is published in [10].

The unique feature of our approach is that the transition function of the continuous process is synthesized automatically. As the synthesized program is composed of ordinary Java methods, it can utilize all the features of the Java platform. Most importantly in this case, synthesized programs are also able to schedule new events. Moreover, events may invoke synthesized code blocks for calculation, similarly to what was shown in Section 4.1.

5. Concluding Remarks

We have complemented the hybrid simulation method with automatic synthesis of programs to facilitate the development of large computer network simulations. Program synthesis adds more flexibility to specifying simulation problems and provides the means for raising the abstraction level where it is beneficial. The paper presented a simplified application example of this hybrid method and described briefly the simulation environment CoCoViLa.

In order to get good performance, the CoCoViLa environment uses compilation instead of interpretation for the simulation program. In essence, a simulation program is synthesized as an attribute evaluator on the attribute model, extracted from the simulation problem specification. In addition, the program may contain event handlers that are used for discrete part of the simulation. Infrastructure to support discrete and hybrid simulations was implemented and integrated into the CoCoViLa environment.

We have not discussed parallel and distributed simulation here. However, the CoCoViLa system that is the kernel of the simulation environment does not prohibit developing Java programs as components that run in a distributed way. There is an older programming environment NUTS [21] that has a similar technique of program construction and is a distributed computing system successfully used for large simulations [9].

Acknowledgment

This work was partially supported by the grant No. 372/0807 of the Estonian Ministry of Defence.

References

[1] The ns-3 network simulator. `http://www.nsnam.org/`, [2/25/2009].

[2] OMNeT++ discrete event simulation system. `http://www.omnetpp.org/`, [2/25/2009].

[3] François Baccelli and Dohy Hong. Flow level simulation of large IP networks. In *IEEE INFOCOM*, 2003.

[4] Stephan Bohacek, João P. Hespanha, Junsoo Lee, and Katia Obraczka. A hybrid systems modeling framework for fast and accurate simulation of data communication networks. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):58–69, 2003.

[5] Laurent Fournié, Dohy Hong, and Florent Perisse. NetScale: scalable time-stepped hybrid simulation of large IP networks. *ACM SIGCOMM Computer Communication Review*, 36(5):35–38, 2006.

[6] Pavel Grigorenko, Ando Saabas, and Enn Tyugu. Visual tool for generative programming. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 249–252, New York, NY, USA, 2005. ACM Press.

[7] Pavel Grigorenko and Enn Tyugu. Deep semantics of visual languages. In E. Tyugu and T. Yamaguchi, editors, *Proceedings of the Seventh Joint Conference on Knowledge-based software engineering*, volume 140 of *Frontiers in Artificial Intelligence and Applications*, pages 83–95. IOS Press, 2006.

[8] G. Grossschmidt and M. Harf. Modelling and simulation of fluid power systems in an intelligent programming environment. In *ISC'08: Proceedings of the 6th International Industrial Simulation Conference*, pages 224–230, Lyon, France, 2008. EUROSIS.

[9] G. Grossschmidt, J. Vanaveski, and M. Harf. Simulation of hydraulic chains using multi-pole models in the NUT programming environment. In *Proceedings of the 14th European Simulation Multiconference on Simulation and Modelling*, pages 709–713. SCS Europe, 2000.

[10] Yu Gu, Yong Liu, and Don Towsley. On integrating fluid models with packet simulation. *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 4:2856–2866, March 2004.

[11] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, and H. Zheng. Ptolemy II – heterogeneous concurrent modeling and design in Java, 2003.

[12] Raj Jain and Shawn A. Routhier. Packet trains – measurement and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4(6), 1986.

[13] Cameron Kiddle, Rob Simmonds, Carey Williamson, and Brian Unger. Hybrid packet/fluid flow network simulation. In *PADS '03: Proceedings of the seventeenth workshop on Parallel and distributed simulation*, page 143, Washington, DC, USA, 2003. IEEE Computer Society.

[14] Hongjoong Kim and Junsoo Lee. Variable step fluid simulation for communication network. In Fernando Boavida, Thomas Plagemann, Burkhard Stiller, Cédric Westphal, and Edmundo Monteiro, editors, *Networking*, volume 3976 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 2006.

[15] Yong Liu, Francesco Lo Presti, Vishal Misra, Don Towsley, and Yu Gu. Fluid models and solutions for large-scale IP networks. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 91–101, New York, NY, USA, 2003. ACM.

[16] Benjamin Melamed, Shuo Pan, and Yorai Wardi. HNS: A streamlined hybrid network simulator. *ACM Transactions on Modeling and Computer Simulation*, 14(3):251–277, 2004.

[17] David M. Nicol and Guanhua Yan. Discrete event fluid modeling of background TCP traffic. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(3):211–250, 2004.

[18] Andres Ojamaa and Enn Tyugu. Rich components of extendable simulation platform. In Hamid R. Arabnia, editor, *MSV*, pages 121–127. CSREA Press, 2007.

[19] George F. Riley, Talal M. Jaafar, and Richard M. Fujimoto. Integrated fluid and packet network simulations. In *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, MASCOTS 2002*, pages 511–518, 2002.

[20] E. Tyugu and R. Valt. Visual programming in NUT. *Journal of visual languages and programming*, 8:523–544, 1997.

[21] V. Vlassov, M. Addibpour, and E. Tyugu. NUTS: a distributed object-oriented platform with high level communication functions. *Computers and Artificial Intelligence*, 17(4):305–335, 1998.

Publication IV

Ojamaa, Andres; Tyugu, Enn (2007).

Rich Components of Extendable Simulation Platform

Proceedings of the 2007 International Conference on Modeling, Simulation & Visualization Methods, MSV 2007: June 25–28 2007, Las Vegas Nevada, USA. Ed. Arabnia, Hamid R. Las Vegas: CSREA Press, 121–127.

Rich Components of Extendable Simulation Platform

Andres Ojamaa
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia

Enn Tyugu
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia

Abstract – *We present a design and implementation of a new type of components of Java based simulation environment — a rich component. It can be considered as an extension of a Java class like the Java bean is. Besides visual and algorithmic parts, it has also a logical part that can be used for guiding the automatic composition of a simulation program. Its algorithmic part includes a daemon that provides flexibility of user interface as well as a convenient way for developing multithreaded simulation programs. A toolbox for rich components has been implemented as a programming environment CoCoViLa.*

Keywords: Simulation, software components, visual specification, Java-based modelers.

1 Introduction

The first object-oriented general purpose programming language Simula-67 [3] was developed as a successor of a simulation language Simula [4], where the idea of the object-oriented software development was present in a nutshell. Since then, the object-oriented approach to simulation software development has prevailed, even before it became a generally applicable technology. However, the general concepts of object and class are insufficient for todays component-based simulation software development. There are two well-known extensions of objects and classes — Java beans and software agents that could be (and to some extent are) used as components of simulation software. Java beans add a visual dimension to components, and they are excellent means for GUI development, but they are not widely used as basic components in simulation. Software agents can possess considerable intelligence and flexibility, but they are rather heavy weight at runtime, and have still a limited usage in simulation. In both cases, considerable overhead is an obstacle of the usage of these components in high-performance applications. In the present work we pursue the extension of objects in the same direction as beans, and propose *rich components* as building blocks of simulation software. Like Java beans or agents, also rich components require special middleware for their usage. However, they are designed in such a way that the middleware acts mainly as a compiler, and does not decrease the runtime performance.

A number of simulation environments, often domain specific, have been developed around the concept of reusable software components. AgentSheets [10] is a commercial visual authoring tool for creating agent based interactive simulations. The environment enables the transformation of simulations to Java applets or Flash movies embeddable in web pages. Visual Simulation Environment (VSE) [1] is another commercial software product intended for developing and executing general purpose discrete-event simulation applications. VSE includes a graphical tool called VSE Editor that lets one to build simulation models using the object-oriented paradigm with inheritance, encapsulation and message passing. Once built, the VSE Simulator is used to make experiments on the simulation model. While AgentSheets and VSE both provide visual development tools and interactive components as basic building blocks of simulation models, the components are missing logical specifications. Without any logical description there are less opportunities for generating parts of the simulation model or program automatically.

There are also many academic projects aiming to develop better solutions for simulation applications. For example, the Ptolemy project that studies modeling, simulation and design of concurrent, real-time, embedded systems has implemented a software framework named Ptolemy II [8]. It is a component assembly framework based on Java. Simulation models are composed of reusable visual library components many of which are data and domain polymorphic. Another approach is taken by the J-Sim project [12] where components are structured according to the autonomous component architecture. This means that software components are developed like modern integrated circuit chips. A finished component is a black box with full specification of input and output signal patterns. Moreover, researchers have

proposed yet different types of reusable components for easy composition of simulations. In the paper [9] a new component architecture is described with the emphasis on the communication abilities of the objects. The shortcomings of the approaches referred in this paragraph are the lack of interactive properties in the first two cases. The last one again does not provide logical specifications for the components.

We are going here to present a software environment that implements the concept of rich components, and is supplied with a collection of specific components of this form for simulation purposes. First, we describe the concept of rich component, thereafter we briefly discuss the toolbox CoCoViLa that supports these components. In the following sections we discuss all parts of rich components in more detail and give examples of their application.

2 Rich components

Rich components are descriptions of domain-specific concepts that are used in simulation. Before going to explain rich components in more detail, we shall briefly explain the context of their usage and outline the principal requirements in this way. Rich components are collected into packages. A package together with the middleware that is CoCoViLa programming environment [5] presents a domain-specific simulation tool. These tools are built and extended just by creating a package and adding new rich components to the package. In particular, a package should include a simulation engine (or several simulation engines in the case of hybrid simulation) that is also presented as a rich component, and it *can be of considerable size*. From the other side, rich components may be very small entities that are used intensively in an internal loop of a simulation program and their usage *must not create significant overhead*. Graphical description of simulation objects is a natural requirement. Hence, rich components must have a *graphical representation*. The next requirement is to have *simple translation from graphical representation into textual representation*. This will guarantee the possibility of debugging on all levels. An important functionality of a rich component is *interaction with a user during the simulation process*. Finally, rich components should *have logical specification* that will enable one to compose a simulation program automatically. These features, including the automatic composition of programs, are supported by CoCoViLa. Considering these requirements, we have designed a rich component in four parts:

- visual part
- logical part (specification part)
- program component
- daemon.

The visual part is for interaction with a user. The logical part and the program component are presented as a *metaclass*, see also [5]. This is a Java class, where the logical part is included as a comment, called a *metainterface*. The daemon is a Java class that describes a thread started by a user, if it is needed. Using daemons enables one to develop flexible interfaces to simulation programs. Not all rich components have to include daemons. Fig. 1 and Fig. 2 show two views of a rich component *Boiler*. The first is a user's view where only visual representation and some informal semantics are included.

The second is a developer's view where all four parts of the rich class are included. We see two Java classes *Boiler* and *BoileDaemon* as well as visual image of the rich class there. The *Boiler* class includes two parts of the rich class: logical part and program component.

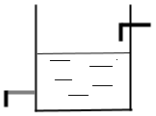
In the following sections we are going to explain the design and usage of all parts of the rich component in more detail.

3 Toolbox

We have developed a toolbox CoCoViLa that is a core platform for simulation packages [5]. It provides tools for developing rich components, constructing simulation packages, specifying simulation problems and running them. From a user's point of view the toolbox consists of two programs: *Class Editor* for the component/package development and *Scheme Editor* for visual specification of simulation problems and for performing the simulation.

3.1 Class Editor

Class Editor supports a language designer in defining visual aspects of concepts, and also their logical and interactive aspects. In our terms — Class Editor is a tool for developing visual parts of rich components and for binding them with other parts developed as Java classes.



inFlow — inflow, liter/min
outFlow — outflow, liter/min
q — quantity of water in the boiler, liters
$\Delta q = inflow - outflow$

The connection points (ports) are endpoints of tubes.
The boiler signals audibly out of limits quantity of water.
...

Figure 1: User's view of a rich class

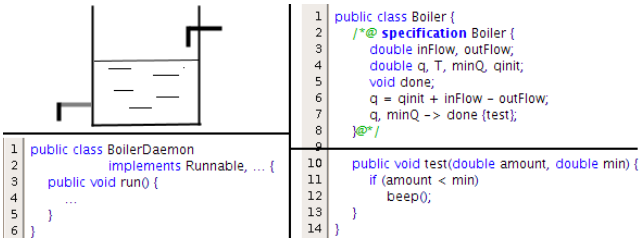


Figure 2: Developer's view of a rich class

Fig. 3 shows the development of a rich component *Adder* in the window of the Class Editor. We can see the image of *Adder* with three ports (connection points) and a pop-up window for defining properties of a highlighted port. Another pop-up window is for defining component properties, in particular, interactive properties of *Adder*, e.g. the fields that will be shown in a dialog window in the Scheme Editor.

3.2 Scheme Editor

Scheme Editor is a tool for performing simulations. It is intended for developing schemes of simulated systems, compiling and running programs. It is used for compiling programs from the schemes according to the specified semantics of a particular domain. It provides an interface for visual programming — putting together a scheme from visual images of concepts. After loading a package into the scheme editor workspace, we get an environment for a particular simulation domain that allows a user to draw, edit and compose schemes through language-specific menus and toolbars. It allows one to compile a simulation program from a given scheme and to control the simulation process through the daemons of rich components.

Fig. 4 shows the scheme editor in use, when a package for simulating simple dynamic systems has been loaded. Such are systems described by ordinary differential equations. The scheme describes an oscillator that has been composed by connecting ports of components of the following types: *Integrator*, *Adder*, *Clock* and *Graph*. The toolbar at the top of the scheme is for adding objects and connections to the scheme. One pop-up window is for instantiating object attributes, another pop-up window is for manipulating the scheme — deleting and arranging objects etc. The scheme editor is fully syntax directed and the correctness of the scheme is forced during editing.

Some words must be said about the compilation of a simulation program. First of all, a scheme is translated into a textual specification that represents the same information that the scheme does. The textual specification language has a precise semantics implemented by means of an attribute technique. We have developed and implemented in CoCoViLa an efficient method of attribute evaluation for specification languages [6]. This is dynamic evaluation of attributes on higher order attribute models that are used for describing the semantics of visual languages.

Fig. 5 shows a textual specification obtained from the scheme of the oscillator shown above. One can see that the specification is created as a logical part of a rich class *Dif*. The figure shows also a simulation algorithm synthesized from the specification. This is algorithm is compiled into a Java class that in the present case is 82 lines and is not shown here. This class can be compiled and run in the CoCoViLa environment. The scheme and the algorithm windows shown in Fig. 5 are useful for

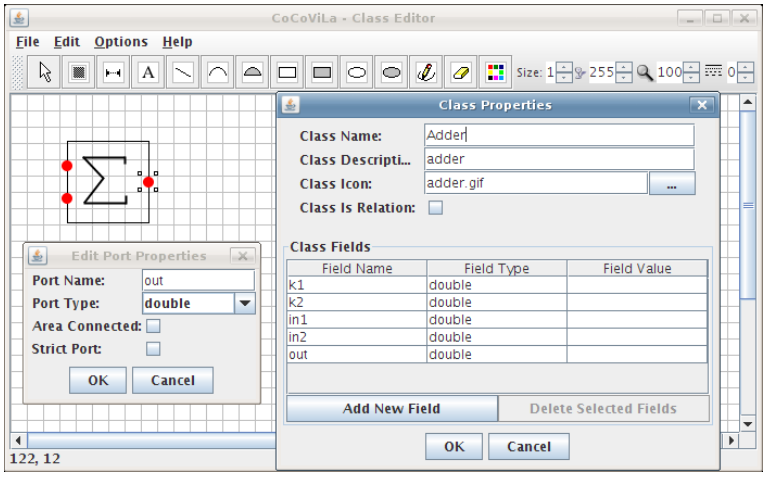


Figure 3: Class Editor window

debugging a specification, but they are not needed when solving simulation problems in CoCoViLa, because the user interacts with a simulation program through the Scheme Editor main window shown in Fig. 4.

4 Visual representation

The visual representation of a rich component consists of several pieces. First, each component has a small image attached that is displayed in the toolbar. The user of the Scheme Editor can create new instances of a specific component by clicking the button decorated with the corresponding icon. Second, the basic look of a scheme object is represented in the package file using XML syntax and simple shapes such as ovals, polygons and lines as well as more advanced features like text elements and transparency. The language used for representing graphics is not very different from a small subset of SVG [2].

Components may also have one or more ports which provide an interface for connecting objects and their attributes to each other. Each port must have a name and a type. Only ports of matching type can be connected. Optionally, a port can have different looks depending on whether it is open or connected to another port. The definitions of the graphical representations of a port and a component in the package file are similar.

Fields of a component can be declared to be editable through the properties window of the Scheme Editor. Like ports, different graphical representations may be defined for known and unknown values of a field. This feature lets the components to provide limited visual feedback about the values of the fields in the Scheme Editor. For more detailed description of the package format we refer to [11].

5 Logical part and program component

Logical part and program component constitute jointly a single Java class as shown in Fig. 2. The logical part is intended for expressing attribute semantics of the model described by means of a rich class or by a composition of rich classes. It is included in the Java class as a comment, another part of the class presents data and methods that can be used as specified in the logical part. The logical part is a specification written in a simple specification language that includes:

- Specifications of variables
  $type\ id, [id, ...]$
- Bindings
  $var1 = var2$
- Axioms
  $precondition \rightarrow postcondition\{implementation\}$

Types can be primitive Java types (`int`, `double` etc), Java classes or rich classes. *Bindings* are used for

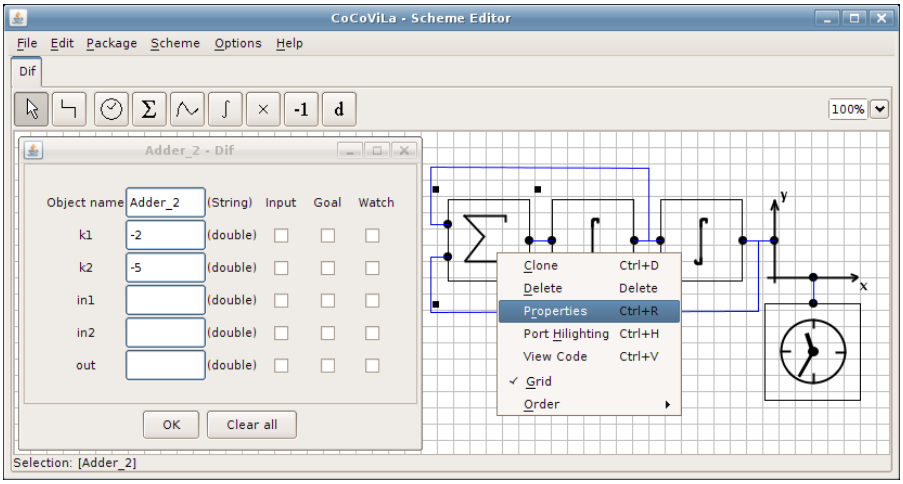


Figure 4: Scheme Editor window

structural composition, to specify the equality of two variables *var*1 and *var*2 (these can be components of other variables declared in the specification).

Axioms are specifications of methods of the class. The *preconditions* of axioms can be conjunctions of propositional variables and implications of conjunctions of propositional variables. The *postconditions* are conjunctions of propositional variables. Name of any variable from specification can be used as a propositional variable, denoting that value of the variable can be computed. An implication in a precondition denotes a goal — a computational problem whose algorithm has to be synthesized before the method with the precondition can be applied. This logic has been tested in several practically applied synthesizers, in particular, in the NUT system [13]. *Implementation* is a name of a method of the class being specified.

For convenience of the user we have included simple algebraic equations in the specification language:

expression = *expression*,

where expressions are arithmetic expressions of a restricted form that depends on the equation solver. An equation can be considered as a specification both of methods and of axioms. The methods represented by an equation are those that solve the equation with respect to its variables.

A specification in this language describes a higher-order attribute model. Higher order means that inputs of attribute dependencies can be functions that must be synthesized on the attribute model. The attribute evaluation on higher order attribute models is described in [6]. This method provides a dynamically composed attribute evaluator for a given attribute model and a given problem in the form "given $x_1,\ldots,x_m$ compute $y_1,\ldots,y_n$". This technique is efficient and reliable. In practice, composition of an attribute evaluator takes a fraction of second.

6 Daemon

The daemon is an optional part of a rich component that is used to make the component interactive. Basically, the daemon is just a class implementing a set of methods with predefined signature. Before the first instantiation of the rich component the framework explores the interface declared by the daemon class through reflection. After the discovery phase the methods that were recognized as belonging to the daemon API are hooked to the framework's internal registry. This registry maps extension points provided by the framework to specific daemon instances. Upon receiving an event which might be for example the result of a GUI action taken by the user, the framework consults the registry to find all daemons interested in being notified about this particular type of events. Then, each of these daemons is notified by calling respective method on the instance.

The set of all possible events a daemon could want to react is large starting from low level events such as mouse movements and ending with high level semantic

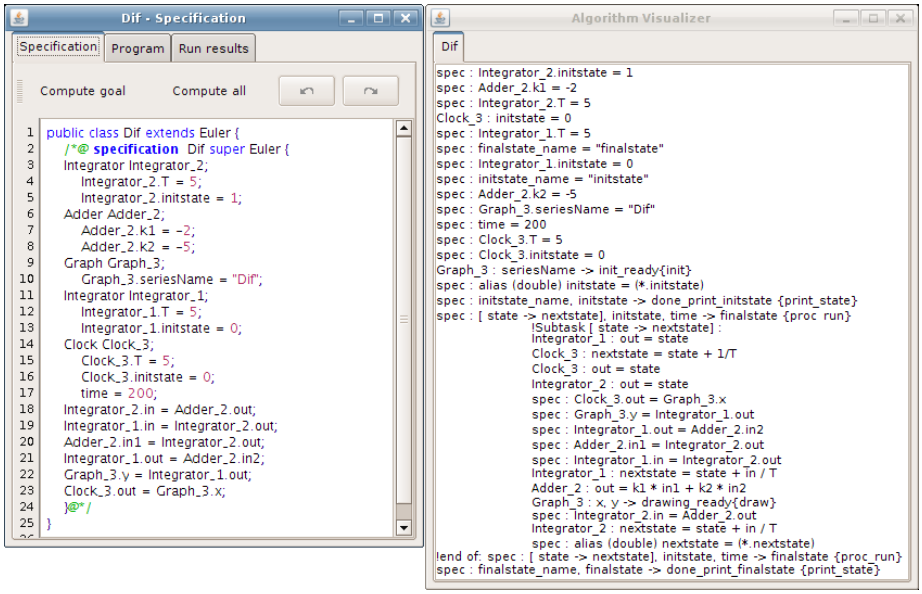


Figure 5: Specification and algorithm of an oscillator

events, for example creation of a new rich component instance. Therefore, it is essential to break the API into small manageable subsets. In this way the developer of a rich component needs only to implement the methods that are really necessary. As a side effect, runtime overhead is kept minimal by avoiding superfluous method calls. Also, the reflective nature of the interface discovery process frees the developer from describing the interface twice.

A daemon can be started in a separate thread to facilitate parallel background processing. This can be achieved by just implementing the *Runnable* interface with the *run* method. The middleware takes care of spawning new threads. In addition, each daemon is given the reference to the scheme being composed by the user. Having the reference to the scheme gives the daemons full control over the scheme. This enables one to develop components that can assist the user in the task of composing a simulation model by displaying dialog windows and arbitrarily complex visual feedback or directly modifying the model. For example, a daemon can be used to enforce some complex syntax rules on the structure of the visual scheme. To restrict connecting incorrect ports the daemon has to implement a boolean returning function that will be called each time the user tries to make a new connection between ports. When the function returns false, the action will be cancelled.

7 Conclusions

We have introduced rich components as a flexible way to represent concepts of simulation domains. A rich component has abstract properties described in a declarative way in its logical part. It has algorithmic properties described by methods of a Java class. If needed, it can have interactive properties described by another Java class whose *run* method runs as a daemon in parallel with the main simulation program. Finally, a rich class has a visual representation whose components (ports) are bound with the variables of the logical part.

In order to get good performance, we have used compilation instead of interpretation for the simulation program. In essence, a simulation program is synthesized as an attribute evaluator on the attribute model, extracted from the simulation problem specification.

We have not discussed parallel and distributed simulation here. However, the CoCoViLa system that is the kernel of the simulation environment does not prohibit developing Java programs as components that run in a distributed way. There is an older programming environment Nuts [14] that has a similar technique of program construction and is a distributed computing system successfully used for large simulations [7].

Acknowledgements

This work has been supported by the grant No. 6886 of the Estonian Science Foundation.

References

[1] Osman Balci, Anders I. Bertelrud, Charles M. Esterbrook, and Richard E. Nance. A picture-based object-oriented visual simulation environment. In *WSC '95: Proceedings of the 27th conference on Winter simulation*, pages 1333–1340, 1995.

[2] World Wide Web Consortium. Scalable Vector Graphics (SVG) 1.1 specification. `http://www.w3.org/TR/SVG11/`, 2003.

[3] Ole-Johan Dahl. *SIMULA 67 common base language, (Norwegian Computing Center. Publication)*. 1968.

[4] Ole-Johan Dahl and Kristen Nygaard. SIMULA: an ALGOL-based simulation language. *Commun. ACM*, 9(9):671–678, 1966.

[5] Pavel Grigorenko, Ando Saabas, and Enn Tyugu. Visual tool for generative programming. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 249–252, New York, NY, USA, 2005. ACM Press.

[6] Pavel Grigorenko and Enn Tyugu. Deep semantics of visual languages. In E. Tyugu and T. Yamaguchi, editors, *Proceedings of the Seventh Joint Conference on Knowledge-based software engineering*, volume 140 of *Frontiers in Artificial Intelligence and Applications*, pages 83–95. IOS Press, 2006.

[7] G. Grossschmidt, J. Vanaveski, and M. Harf. Simulation of hydraulic chains using multi-pole models in the NUT programming environment. In *Proceedings of the 14th European Simulation Multiconference on Simulation and Modelling*, pages 709–713. SCS Europe, 2000.

[8] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, and H. Zheng. Ptolemy II – heterogeneous concurrent modeling and design in Java, 2003.

[9] Oray Kulac and Murat Gunal. Combat modeling by using simulation components. Technical report, Turkish Naval HQs R&D and Decision Support Department, 2002.

[10] Alexander Repenning, Andri Ioannidou, and John Zola. AgentSheets: End-user programmable simulations. *J. Artificial Societies and Social Simulation*, 3(3), 2000.

[11] Ando Saabas. A framework for design and implementation of visual languages. Master's thesis, Tallinn University of Technology, 2004.

[12] Ahmed Sobeih, Wei-Peng Chen, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Hung-Ying Tyan, and Honghai Zhang. J-Sim: A simulation environment for wireless sensor networks. In *ANSS '05: Proceedings of the 38th annual Symposium on Simulation*, pages 175–187, Washington, DC, USA, 2005. IEEE Computer Society.

[13] E. Tyugu and R. Valt. Visual programming in NUT. *Journal of visual languages and programming*, 8:523–544, 1997.

[14] V. Vlassov, M. Addibpour, and E. Tyugu. NUTS: a distributed object-oriented platform with high level communication functions. *Computers and Artificial Intelligence*, 17(4):305–335, 1998.

Publication V

Ojamaa, Andres; Haav, Hele-Mai; Penjam, Jaan (2015).

Semi-Automated Generation of DSL Meta Models from Formal Domain Ontologies

Model and Data Engineering: 5th International Conference, MEDI 2015, Rhodes, Greece, September 26–28, 2015, Proceedings. Ed. Bellatreche, Ladjel; Manolopoulos, Yannis. Springer, 3–15. (Lecture Notes in Computer Science; 9344).

Semi-automated Generation of DSL Meta Models from Formal Domain Ontologies

Andres Ojamaa, Hele-Mai Haav(✉), and Jaan Penjam

Institute of Cybernetics, Laboratory of Software Science,
Tallinn University of Technology, Tallinn, Estonia
{andres.ojamaa,helemai,jaan}@cs.ioc.ee

Abstract. This paper addresses the problem of alignment of domain ontologies and meta-models of Domain Specific Languages (DSL) in order to facilitate the DSL development process by formal methods. The solution presented in this paper automatically generates design templates of a DSL meta-model that are consistent with a given domain ontology represented in OWL DL. Consistency of alignment is ensured by predefined mapping rules between constructs of ontology modelling language OWL DL and a modelling language used for representing DSL meta-models. The approach is implemented as an extension to the CoCoViLa system and the CoCoViLa modelling language is used for representing DSL meta-models. The evaluation of the provided method is carried out by developing the DSL for the IT risk analysis and management domain.

Keywords: Model-driven software engineering · Ontology-based modelling · Model transformations · DSL meta-models

1 Introduction

In recent years, several approaches to incorporate ontologies into general frameworks of Model Driven Software Engineering (MDSE) have been proposed [1, 9, 16, 19]. In addition, an effort is put to using ontologies in the field of DSL engineering [5, 17–20]. However, in order to facilitate the DSL development process by utilization of formal methods more attention needs to be paid to alignment of domain ontologies as formal domain models with DSL meta-models. This creates a new challenging task for software engineers not well supported by existing traditional and ontology-driven MDSE methods. This paper provides a method and tools to perform this task.

The main contribution of the paper is an approach that focuses to partial automation of the design and implementation phases of the DSL development process by introduction of formal domain ontologies into this process and automatic generation of design templates of a DSL meta-model from a given domain ontology. Ontology Web Language (OWL) [14] is used for representing formal domain ontologies. The approach is implemented as an extension to the CoCoViLa system [10] and the CoCoViLa modelling language is used for representing meta-models of DSLs. The CoCoViLa system enables automatic generation of executable Java programs according to the given DSL meta-model. The evaluation of the provided method is carried out by developing the DSL for the IT risk analysis domain.

L. Bellatreche and Y. Manolopoulos (Eds.): MEDI 2015, LNCS 9344, pp. 3–15, 2015.
DOI: 10.1007/978-3-319-23781-7_1

Novelty of our approach comparing to other ontology-driven software development methodologies lies in using formal domain ontologies as a basis for automatic generation of design templates of a DSL meta-model that are consistent with the given domain ontology. This makes a DSL closely aligned with the domain for what it is designed for. Consistency of alignment is ensured by predefined mapping rules between constructs of the ontology modelling language OWL and a modelling language for DSL meta-models (i.e. the CoCoViLa modelling language in our case).

The development of this new approach was motivated by practical needs of using the CoCoViLa modelling tool for the development of different DSLs in the domains of simulation of hydraulic systems as well as simulations of security measures for banking and communication networks [10]. From these experiences we have learned that tighter integration of domain knowledge with the corresponding DSL will create advantages in achieving consistency of the DSL with domain knowledge and will ease maintenance of applications developed by using the DSL.

The rest of the paper is structured as follows. Section 2 is devoted to related work and Sect. 3 provides an overview of the CoCoViLa modelling language and the system giving background knowledge for our approach. In Sect. 4, our approach for domain ontology driven DSL engineering is presented. Section 5 is devoted to the evaluation of the provided method by implementation of a prototype of the DSL for the IT security risk analysis and management field. Section 6 concludes the paper.

2 Related Work

Ontologies in computer science represent computer-usable specifications of basic concepts in a domain and relationships among them. Ontologies are usually expressed in a logic-based language. There are several ontology languages available, but the most widely used are the W3C standards OWL DL and OWL2 [14].

In software engineering, domain analysis plays an important role in understanding the domain of interest. Nowadays, many researchers propose to use ontology engineering methods in the domain analysis process [1, 4, 19, 20] and some of them suggest to represent resulting model of domain analysis by domain ontology [4, 7, 20]. In these works, domain models are considered as descriptive models consisting of a set of domain instances (ABox in Description Logics (DL) [2]) and a set of classes for classifying these instances (TBox in DL). Comparing to traditional analysis models formal ontologies have additional useful features like reasoning based on DL [2].

Most of existing ontology driven MDSE methodologies use ontological services for performing model consistency checking and model transformations [16, 19]. Some of the approaches use domain ontologies (e.g. represented in UML) as a part of Computation Independent Models (CIM) [18]. Few works address also semantic search and composition of models of software components [9].

Several proposals [1, 19, 20] are about integrating ontologies to the OMG meta-pyramid of Model Driven Architecture (MDA) [15]. For example, in [1] ontology-aware mega-model is provided for ontology integration to MDA. Another approach in [19] shows that the Ecore meta-meta-model of the Eclipse Modelling Framework [6] and OWL2 meta-model can be integrated in order to provide a

meta-meta-model for modelling DSLs. In [20] OWL2 ontologies are integrated into the meta-meta-model level in order to support joint DSL and domain engineering.

Similarly to these works, we also introduce ontologies to the meta-meta-model level but we are focused to partial automation of creation of DSL meta-models.

There are frameworks that examine domain ontologies as domain models that can be automatically transformed to a DSL grammar [4, 5]. We were inspired by these works but our approach uses predefined mapping rules.

3 The CoCoViLa Modelling Language and System Overview

The approach presented in this paper is developed as an extension to the CoCoViLa modelling system[1] that is implemented in Java. The CoCoViLa modelling language consists of visual and textual declarative languages for developing DSLs for engineering fields, where scientific and engineering computations play a crucial role. Intended users of the CoCoViLa modelling language are DSL designers (together with domain experts), who create meta-models and a DSL for a particular domain. DSL designers can create textual and/or visual DSLs.

The most important construct of the textual declarative modelling language is a concept specification that represents a collection of instances. Concept specifications can be arranged into taxonomy. Concept specifications include descriptions of structural components of a concept as declarations of variables. In addition, they may include relations that are specifications showing how to derive values of some variables from the values of other variables. Relations are divided to equations and axioms. Equations define dependencies between variables bound by the equation. Axioms describe functional dependencies between variables and they differ from equations in that they have realizations as Java methods.

Important feature of the modelling language is its grounding with a subset of Intuitionistic Propositional Calculus (IPC) [13] that is used as a logical language for representation of domain specific axioms. From the given concept specifications and the task specification the CoCoViLa system automatically constructs the algorithm of the program and generates the Java program that solves the computational problem given by the task specification. The latter is a statement of a computational problem that specifies what outputs are to be computed from given inputs. It does not have a given realization but its realization is attempted to generate automatically from the constructive proof of the corresponding theorem in IPC according to the inference rules of Structural Synthesis of Programs (SSP) [12].

When using the CoCoViLa modelling language we distinguish between the DSL meta-modelling and application specific modelling levels. In our terminology, a DSL meta-model consists of a set of concept specifications and an application specific model consists of a task specification and a set of valuations of variables. We explain the main idea behind the language using a simplistic example from the geometry domain. The geometry DSL may contain specifications of the `Square` and the `Circle` concepts including all necessary variables and equations. If we are interested in calculating the

[1] http://cocovila.github.io/.

area that is difference of areas of a square and a circle where diagonal of a square is equal to diameter of a circle, then we need to create a new concept specification (e.g. `SquareWithinCircle`) as depicted in the following Fig. 1.

```
specification SquareWithinCircle {
            double area_difference;
            Square S;
            Circle C;
            C.diameter=S.diagonal;
            area_difference = C.area-S.area;}
```

Fig. 1. An example of the CoCoViLa modelling language

Dot notation is used to refer to inner concept specifications. As a result, the geometry DSL is composed from the following concept specifications: `Square`, `Circle`, and `SquareWithinCircle`.

On the application specific modelling level, we can use this DSL and specify different task specifications (e.g. `->area_difference`), which specify computational problems that need to be solved. Values of inputs are assumed to be given (e.g. `S.hasHeight = 10`) or computable on the basis of the specification. If output variables of a task are computable from (possibly empty) list of inputs, the task is solved and the Java program is automatically generated. Besides equations, a Java method could be declared to be as a realization of a given functional dependency. For example, instead of the equation `area_difference = C.area-S.area` we may write axiom `C.area, S.area ->area_difference {<JavaMethod>}`.

The syntax of the full CoCoViLa modelling language is presented in [10]. Using the current system, DSL meta-models are created manually by DSL developers. In practice, informal methods are used for representation of domain models.

4 An Approach to Domain Ontology-Driven DSL Engineering

4.1 Introducing Formal Ontologies to the DSL Development Process

For improvement of the workflow of DSL development we introduce two types of ontologies into the DSL development process: domain and system ontologies.

Purpose of *domain ontology* is to provide specification of conceptualization of domain knowledge. Formal domain ontology could be seen as a static part of a meta-model of a DSL, particularly a part of a CIM. Using formal ontologies as a part of a CIM makes connections from CIM to Platform Independent Model (PIM) transparent and methodical. We automate creation and maintenance of connections between CIM and PIM by automatic generation of design templates of DSL meta-models from formal domain ontologies. In our case (i.e. the CoCoViLa system extension), design templates of a DSL meta-model are concept specifications that do not include specifications of dynamic parts of a meta-model like axioms and equations (except binding, e.g. see Table 1). These need to be added manually to the template.

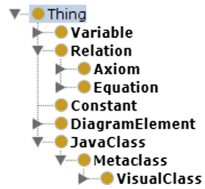


Fig. 2. A fragment of system ontology (a screen shot of the Protégé ontology editor)

Purpose of *system ontology* is formally describing the modelling language and the system concepts (see Fig. 2 for a part of a class hierarchy).

We utilize the CoCoViLa system ontology in the implementation phase of a DSL. Individuals of the system ontology classes together with their property values are used to store knowledge about a particular DSL meta-model. It is important to notice that domain ontology and the system ontology are linked to each other for this purpose.

4.2 Mapping Formal Domain Ontologies to DSL Meta-Models

For automatic transformation of formal domain ontologies to DSL meta-models corresponding mappings between a set of ontology representation language constructs and a DSL modelling language constructs need to be defined. As depicted in Fig. 3, in our case we need to provide mappings between a subset of OWL DL constructs and a subset of the CoCoViLa modelling language constructs.

Both languages are declarative languages intended to be used for knowledge representation. OWL DL semantics is given by DL [2] and the CoCoViLa modelling language semantics is based on a subset of IPC [13]. DL enables to reason whether domain ontology is consistent and complete. Expressive power of SSP used for deciding about computational correctness of a program automatically generated from a given specification is equivalent to IPC [13].

Since semantical basis of both languages is different, then some restrictive conditions must be placed on domain ontology structure in order to ensure that it can be properly transformed into a set of constructs of the CoCoViLa modelling language. These restrictions mainly concern OWL constructs for object property characteristics, property restrictions and complex classes that cannot be mapped to the CoCoViLa language.

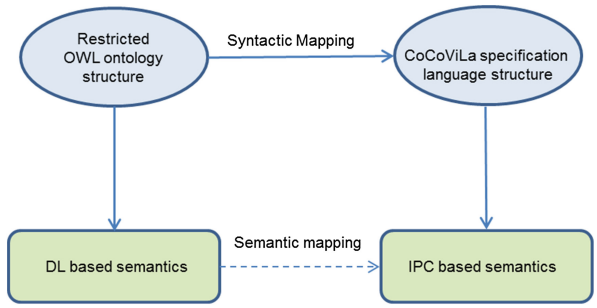


Fig. 3. Mapping formal domain ontology to the CoCoViLa language

Table 1. Mappings between OWL and the CoCoViLa modelling language constructs

Acceptable OWL constructs of formal domain ontology (i.e. the CoCoViLa compatible OWL ontology)	The CoCoViLa modelling language statements generated from corresponding OWL constructs

A class is a set of individuals with similar properties. Classes should be disjoint.
 	
`owl:Class`
`e.g.`
`<owl:Class rdf:ID="Square"/>`
`</owl:Class>`
	A concept specification is a specification of conceptualization of a domain represented in the CoCoViLa textual modelling language. A concept specification represents a collection of instances.
	
	`specification` `<name> {}`
	`e.g.`
	`specification` `Square {}`

Datatype properties represent relations between instances of classes and RDF literals and XML Schema datatypes
 	
`owl:DataTypeProperty,`
`rdfs:domain, XML Schema datatypes`
`e.g.`
`<owl:DatatypeProperty`
`rdf:ID="hasHeight">`
`<rdfs:domain`
`rdf:resource="#Square" />`
`<rdfs:range`
`rdf:resource="&xsd;double"/>`
`</owl:DatatypeProperty>`
	Each concept specification has an associated collection of attributes (represented by concept specification variables in textual specification) which can be filled by values of a given Java type or other concept specifications.
	
	`<Java type> < attribute name>`
	`e.g.`
	`specification` `Square`
	`{`**`double`** `hasHeight;}`

Object properties represent relationships between instances of two classes.
 	
`owl:ObjectProperty`
`e.g.`
`<owl:ObjectProperty`
`rdf:ID="compS"> <rdfs:domain`
`rdf:resource="#SquareWithinCir`
`cle"/> <rdfs:range`
`rdf:resource="#Square"/> …`
	Attributes associated with concept specification can be filled by a concept specification.
	
	`<concept specification name>`
	`<attribute name>`
	`e.g.`
	`specification`
	`SquareWithinCircle {`
	`Square compS;}`

(*Continued*)

Table 1. (*Continued.*)

Two properties may be stated to be the same by using property equivalence.
 	
`owl:equivalentProperty`
`e.g.`
`rdf:about="#hasHeight">`
`<rdfs:domain`
`rdf:resource="#Square"/>`
`<owl:equivalentProperty`
`rdf:resource="#hasWidth"/>`
`<rdfs:range`
`rdf:resource="&xsd;double"/>`
	A binding represents an equality relation (equation) between two variables. It could be used for representation of data and object property equivalences.
	
	`<variable name> = < variable name>`
	`e.g.`
	`specification` `Square {`
	`double` `hasHeight;`
	`double` `hasWidth;`
	`hasHeight=hasWidth; }`

Taxonomic constructor of class hierarchies.
 	
`rdfs:subClassOf`
`e.g.`
`<owl:Class rdf:ID="Square">`
`<rdfs:subClassOf`
`rdf:resource="#Rectangle" />`
`...`
`</owl:Class>`
	The assertion of concept specification taxonomy. Concept specification associated with a Java class B can inherit concept specification of class associated with a Java class A iff A is a superclass of B in Java, i.e. B is declared as class B extends A. No multiple inheritances are allowed.
	
	`specification` `Square` **`super`** `Rectangle {… }`

Individuals with values of datatype properties.
 	
`<owl:NamedIndividual`
`rdf:about="#MySquare">`
`<rdf:type`
`rdf:resource="#Square"/>`
`<hasHeight`
`rdf:datatype="&xsd;double">5.0`
`</hasHeight>`
`</owl:NamedIndividual>`
	The CoCoViLa modelling language does not distinguish between classes and individuals on the specification level. Instances will be created during the code generation of the program for specific problem solving as Java class instances.
	The created instance will be initialized by a declared value.
	
	`<variable name> = <value>`
	`e.g.`
	`MySquare.hasHeight = 5.0;`

Semantic correspondence of mappings is given only indirectly via syntactic mappings in Table 1. Providing semantic mappings and a proof of the corresponding theorem are out of scope of this paper. In Table 1, OWL constructs that are allowed in the CoCoViLa compatible ontologies are listed and mapping rules are provided. The given mappings are used to automatically generate design templates of a DSL meta-model. OWL constructs for what mappings are not defined are not allowed and the corresponding OWL constructs are ignored in the automated model generation process.

4.3 Implementation of the Approach

The approach takes into account important DSL development phases as follows: domain analysis, design, implementation, deployment, testing and maintenance [11]. We suggest using the iterative (agile) DSL development process that starts with building a core of domain ontology by taking into account requirements of domain and a DSL. Domain ontology is created according to an expert centric agile ontology development methodology [8]. DSL design and implementation stages are supported by the tool that is an extension to the existing CoCoViLa system and is also implemented in Java. The implementation is designed to fit into general architecture of the CoCoViLa system. An overall view of the structure of the extension is provided in Fig. 4.

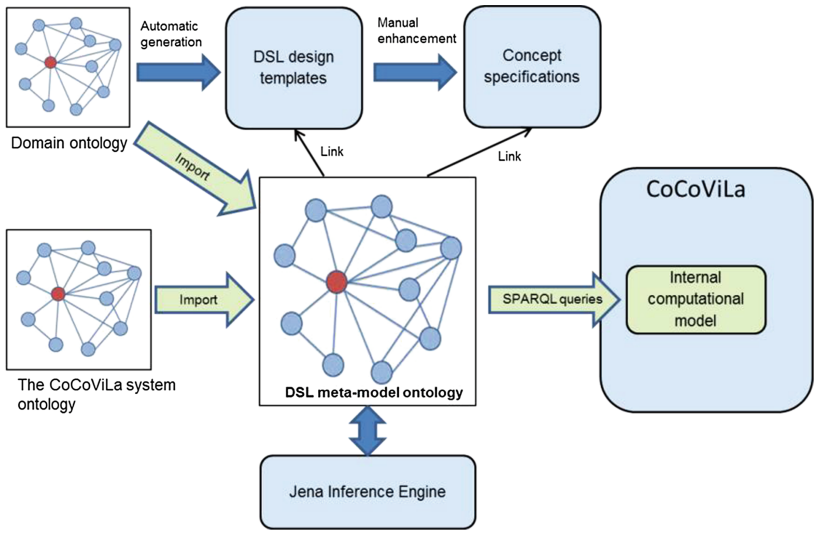


Fig. 4. A general view of the implementation of the approach

Automatically generated templates of a DSL meta-model are implemented as Java classes. For each concept specification from a template, the corresponding Java class is generated. Java classes including concept specifications can be manually enhanced with necessary equations and axioms. In addition, these Java classes may contain Java methods that are the realizations of axioms described in the concept specification. Currently, for each DSL its meta-model ontology is created that imports the CoCoViLa system and domain ontologies. It also contains the DSL specific instances (ABox). Consistency of the DSL meta-model ontology is checked by using OWL DL inference provided by Apache Jena.[2] When loading the DSL, its meta-model ontology is dynamically loaded and SPARQL[3] queries are used to find definitions of elements of

[2] https://jena.apache.org/.

[3] http://www.w3.org/TR/sparql11-query/.

the diagrammatical language and other DSL components in order to insert information about them to the internal computational model of CoCoViLa. After that, the visual part of the DSL is shown in the CoCoViLa DSL window and the DSL is ready to be used by application developers.

5 Evaluation

5.1 A Problem: IT Security Risk Analysis

In order to solve IT security risk analysis problems, a threat modelling tool could be useful. One of the ways to build a threat modelling tool is to develop a DSL for creation of attack simulations using ontological modelling of security knowledge, dynamic attack tree generation techniques and probabilistic models of threat agent behavior. In order to build a particular threat modelling tool following the DSL development approach presented in this paper, we adapted the multi-parameter attack tree method proposed in [3]. Attack trees according to this method are used to estimate the cost and the success probability of attacks. Elementary game theory is used to decide whether the system under protection is a realistic target for gain-oriented attackers.

In the following example that is adapted from [3] a threat analysis for forestalling release is considered. This threat is related to the situation where a competitor of an IT company steals the developed source code and completes it to own product.

5.2 Development of a DSL for the IT Security Risk Analysis Domain

A visual DSL was developed for IT security risk analysis with attack trees methodology. The DSL consists of components to model attack trees and to perform computations on the trees. An attack tree is basically an AND-OR tree that consists of three types of components (nodes) – leaves (atomic threats where there are estimated values for attack parameters), AND-nodes representing complex attacks that are considered successful when all sub-trees are successful, and OR-nodes representing complex attacks that are considered successful when any of the sub-attacks is successful. In addition to the components used for specifying attack trees, there are two additional components required as follows: a simulator component that performs computations on the attack tree models and a visualized component for displaying results.

Formal Ontology of IT Risk Analysis for the Forestalling Release Domain. There are several events necessary for a forestalling release that we consider as domain knowledge. By the attack analysis method that we use, an attack is seen as a game played by the rational attackers and the game is to be profitable for them. In order to decide about the profitability, there are several characteristics related to threat events to be taken into account by the attack game. These constitute domain knowledge related to the particular threat analysis method. Both types of domain knowledge are captured in formal domain ontology of IT security risk analysis. As a result, domain ontology of the forestalling release domain consists of taxonomy of disjoint classes of threat events.

Threat events have associated characteristics needed for the attack game. These are described by data properties which domain is the `Threat_event` class. Main direct subclasses of the `Threat_event` class and data properties that are inherited by its subclasses are shown in Fig. 5.

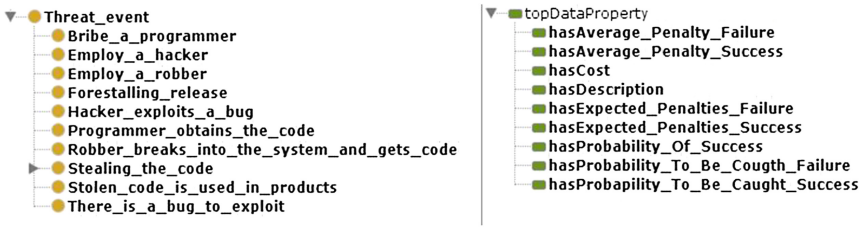


Fig. 5. The forestalling release domain ontology (a screen shot of the Protégé ontology editor)

Generation of DSL Design Templates. According to the given domain ontology and the transformations given in Table 1, the DSL design templates are automatically generated (see a fragment of automatically generated code in Fig. 6).

```
specification Threat_event {
                        string hasDescription;
                        double
                        hasAverage_Penalty_Failure,hasCost,
                        …; }
specification Bribe_a_programmer super Threat_event { }
specification Forestalling_release super Threat_event { }
specification Stealing_the_code super Threat_event { }
…
```

Fig. 6. A fragment of generated DSL meta-model in the CoCoViLa modelling language

Enhancement of DSL Design Templates and DSL Meta-model Ontology. The DSL implementation involves an enhancement of design templates, if necessary. For example, equations used for calculating parameters of an attack game are added to the `Threat_event` specification.

In the DSL meta-model ontology, `Threat_event` is declared as a subclass of the class `Metaclass` from the system ontology. For each subclass of the `Threat_event` class an instance and links to the corresponding `VisualClass` instances are created for capturing diagrammatical part of the DSL. Instances of the `VisualClass` class become components of the attack tree diagram. The DSL is implemented so that the same threat analysis method can be used for different IT risk analysis domains by importing different domain ontologies to the DSL meta-model ontology. The DSL will then be automatically aligned with new domain ontology.

5.3 Analysis of the Evaluation of the Approach

According to the evaluation of the approach by developing a DSL for the IT security risk analysis domain, we identify the following advantages of the approach:

- Since domain analysis is done by developing formal domain ontology, then DL reasoning services can be used for validation of domain models.
- Alignment of formal domain ontologies with DSL meta-models makes it possible to automatically propagate changes in ontology to a DSL i.e. capture the evolution of a domain. It also allows detecting and avoiding errors.
- Formal consistency checking of domain knowledge and a DSL meta-model ontology using DL inference. This is also useful for debugging DSL meta-models. In addition, other resources (e.g. images, multi-media, linked data etc.) could be linked to a DSL meta-model and used as components of a DSL.
- Separation of different kinds of knowledge about the system, domain and a DSL into modular OWL ontologies makes the knowledge more reusable.

However, as our approach is implemented as an extension to the existing CoCoViLa system only partial automation of mapping domain ontology to a DSL meta-model can be provided. Another issue is related to productivity of the DSL development process. We did not provide any productivity analysis yet. We think that there might be problems of creation of formal domain ontologies as ontology engineering techniques do not constitute a part of existing traditional software development methodologies. This may create initial complexity.

6 Conclusion

In this paper, we presented an approach that introduces formal domain ontologies into the DSL development process and allows to automatically generate design templates of a DSL meta-model that are consistent with a given domain ontology represented in OWL DL. The approach was implemented as an extension to the CoCoViLa system. The provided method was tested by developing the DSL for IT risk analysis domain.

Our approach creates the following benefits: formal consistency checking of domain knowledge and a DSL meta-model ontology, automatic generation of design templates of a DSL meta-model and capture of evolution of the domain in a DSL.

Our future work will be related to integrating rules represented in Semantic Web Rule Language (SWRL)[4] to the framework. Rules combined with ontology will allow us to model behavioral aspects (e.g. equations) of a domain and perform corresponding transformations from SWRL to the CoCoViLa modelling language.

Acknowledgements. This research was supported by Estonian Research Council institutional research grant no. IUT33-13, and by the ERDF through the ITC project MBJSDT and Estonian national CoE project EXCS.

[4] http://www.w3.org/Submission/SWRL/.

References

1. Aßmann, U., Zschaler, S.: Ontologies, meta-models, and the model-driven paradigm. In: Calero, C., Ruiz, F., Piattini, M. (eds.) Ontologies for Software Engineering and Software Technology, pp. 249–273. Springer, Heidelberg (2006)
2. Baader, F., Calvanese, D., McGuiness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
3. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational choice of security measures via multi-parameter attack trees. In: López, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 235–248. Springer, Heidelberg (2006)
4. Čeh, I., Črepinšek, M., Kosar, T., Mernik, M.: Ontology driven development of domain-specific languages. ComSIS **8**(2), 317–342 (2011)
5. Fonseca, J.M.S., Pereira, M.J.V., Henriques, P.R.: Converting ontologies into DSLs. In: Pereira, M.J.V., Leal, J.P., Simões, A. (eds.) 3rd Symposium on Languages, Applications and Technologies (SLATE'14), pp. 85–92. Dagstuhl Publishing, Germany (2014)
6. Gronback, R.: Eclipse Modeling Project: a Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional, Boston (2009)
7. Guizzardi, G.: Ontology-based evaluation and design of visual conceptual modelling languages. In: Reinhartz-Berger, I., Sturm, A., Clark, T., Bettin, J., Cohe, S. (eds.) Domain Engineering. Product Lines, Languages, and Conceptual Models, pp. 317–347. Springer, Heidelberg (2013)
8. Haav, H.-M.: A practical methodology for development of a network of e-government domain ontologies. In: Skersys, T., Butleris, R., Nemuraite, L., Suomi, R. (eds.) Building the e-World Ecosystem. IFIP AICT, vol. 353, pp. 1–13. Springer, Heidelberg (2011)
9. Katasanov, A.: Ontology-driven software engineering: beyond model checking and transformations. Int. J. Semant. Comput. **06**, 205–242 (2012)
10. Kotkas, V., Ojamaa, A., Grigorenko, P., Maigre, R., Harf, M., Tyugu, E.: CoCoViLa as a multifunctional simulation platform. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011), pp. 198–205. ICST, Brussels (2011)
11. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
12. Mints, G., Tyugu, E.: Justification of the structural synthesis of programs. Sci. Comput. Program. **2**(3), 215–240 (1982)
13. Mints, G., Tyugu, E.: Propositional logic programming and the PRIZ system. J. Log. Program. **9**(2&3), 179–193 (1990)
14. Motik, B., Patel-Schneider, P.F., Horrocks, I.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. http://www.w3.org/TR/owl2-syntax
15. OMG. MDA Guide 1.0.1. http://www.omg.org/mda June 2003
16. Roser, S., Bauer, B.: Automatic generation and evolution of model transformations using ontology engineering space. In: Spaccapietra, S., Pan, J.Z., Thiran, P., Halpin, T., Staab, S., Svatek, V., Shvaiko, P., Roddick, J. (eds.) Journal on Data Semantics XI. LNCS, vol. 5383, pp. 32–64. Springer, Heidelberg (2008)
17. Tairas, R., Mernik, M., Gray, J.: Using ontologies in the domain analysis of domain-specific languages. In: Chaudron, M.R.V. (ed.) MODELS 2008. LNCS, vol. 5421, pp. 332–342. Springer, Heidelberg (2009)

18. Vanden Bossche, M., Ross, P., MacLarty, I., Van Nuffelen, B., Pelov, N.: Ontology driven software engineering for real life applications. In: Proceedings of the 3rd International Workshop on Semantic Web Enabled Software Engineering, Innsbruck, Austria (2007)
19. Walter, T., Parreiras, F.S., Staab, S.: An ontology-based framework for domain-specific modeling. Softw. Syst. Model. **13**, 83–108 (2014)
20. Walter, T., Parreiras, F.S., Staab, S., Ebert, J.: Joint language and domain engineering. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 321–336. Springer, Heidelberg (2010)

Publication VI

Haav, Hele-Mai; Ojamaa, Andres; Grigorenko, Pavel; Kotkas, Vahur (2015).

Ontology-Based Integration of Software Artefacts for DSL Development

On the Move to Meaningful Internet Systems: OTM 2015 Workshops: Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, EI2N, FBM, INBAST, ISDE, META4eS, and MSC 2015, Rhodes, Greece, October 26–30, 2015, Proceedings. Ed. Ciuciu, I. et al. Cham: Springer, 309–318. (Lecture Notes in Computer Science; 9416).

Ontology-Based Integration of Software Artefacts for DSL Development

Hele-Mai Haav(✉), Andres Ojamaa, Pavel Grigorenko, and Vahur Kotkas

Laboratory of Software Science,
Institute of Cybernetics at Tallinn University of Technology, Tallinn, Estonia
{helemai,andres.ojamaa,pavelg,vahur}@cs.ioc.ee

Abstract. This paper addresses a high level semantic integration of software artefacts for the development of Domain Specific Languages (DSL). The solution presented in the paper utilizes a concept of DSL meta-model ontology that is defined in the paper as consisting of a system ontology linked to one or more domain ontologies. It enables dynamic semantic integration of software artefacts for the composition of a DSL meta-model. The approach is prototypically implemented in Java as an extension to the DSL development tool CoCoViLa.

Keywords: Semantic interoperability · Semantic integration · Ontology-based modelling · DSL meta-models · DSL development

1 Introduction

Domain Specific Languages (DSLs) have been used for a long time in order to shorten the software development lifecycle and make it cost effective in a particular domain of interest. There are many well-known DSLs available like XML for describing data, HTML to mark-up web documents, Structured Query Language (SQL) for querying relational databases, etc. There are also DSLs that are more specific like WebDSL [4] and the Modelica modelling language [2]. In order to create a DSL, several tools have been developed. For example, Xtext[1], MS Visual Studio[2], Metaedit[3], and CoCoViLa[4] are tools that enable the development of a DSL. However, issues related to the integration of these tools and the corresponding DSL meta-models are not entirely solved.

Currently, tools are in many cases integrated on the basis of XML or UML profile SysML or via given transformations of different software artefacts related to DSL models so that models can be imported or exported among tools. However, these representations of software artefacts or corresponding metadata do not formally and explicitly capture semantics of described artefacts. Although model transformations represent semantics, it is encoded into the set of transformation rules.

[1] http://www.eclipse.org/Xtext
[2] https://www.visualstudio.com/
[3] http://www.metacase.com/mep/
[4] http://cocovila.github.io/

I. Ciuciu et al. (Eds.): OTM 2015 Workshops, LNCS 9416, pp. 309–318, 2015.
DOI: 10.1007/978-3-319-26138-6_34

In this paper, we provide a new approach to integration of software artefacts for DSL development using the semantic representation of software artefacts in the form of linked formal ontologies described in Ontology Web Language (OWL) [9]. The role of OWL is to serve as a common language for representation of semantics of software artefacts.

Novelty of our approach is twofold: the semantic integration of distributed software artefacts into a coherent DSL meta-model as well as the simplification of the development lifecycle and evolution of a DSL. In addition, our method also facilitates the semantic integration between DSL meta-models created by different DSL development tools in the case there exists a commitment by software developers to use a common top level system ontology or to explicitly define and make available the system ontology of their tool.

The approach presented in this paper is prototypically implemented as an extension to the DSL development tool CoCoViLa [7]. It allows an automatic generation of executable Java programs according to a DSL meta-model and a specification of an application expressed in the corresponding DSL. In this paper and in CoCoViLa, the term DSL is used to denote a specific type of DSLs i.e. Domain Specific Modelling Languages. However, our approach is general enough to be applied for the development of different kinds of DSLs.

The rest of the paper is structured as follows. Section 2 is devoted to the related work and Section 3 provides background knowledge about the DSL development process with CoCoViLa. In Section 4, our new approach for semantic integration of software artefacts is presented. Section 5 provides an overview of a system architecture supporting the approach. Section 6 concludes the paper.

2 Related Work

Ontologies are used in the existing ontology driven software engineering methodologies in several ways. In general, they are mostly used for the consistency checking of software models and as tools for representing model transformations [13, 14]. Another trend is to integrate ontologies to the OMG meta-pyramid of MDA [15] and to the Ecore meta-meta-model of the Eclipse Modelling Framework [5] in order to provide meta-meta-model for modelling DSL languages [14, 16].

Research that is tightly related to the approach provided in this paper express two views. From the modelling point of view, semantic search and composition of models of software components [6] are important. From the technical point of view, integration of OWL and Java is essential. For example, a hybrid modelling approach that enables software models partially developed in Java and in OWL is given in [12].

However, to the best of our knowledge we do not know the DSL development approaches that use linked ontologies for semantic integration of software artefacts and for dynamic building of DSL meta-models as presented in this paper.

3 Background: the DSL Development Process with CoCoViLa

The CoCoViLa system supports the CoCoViLa modelling language that consists of visual and declarative languages for developing DSLs for domains, where scientific and engineering computations play an important role. Expected users of the CoCoViLa modelling language are DSL designers, who create a meta-model of a DSL for a particular domain. The syntax of the full CoCoViLa modelling language is presented in [3]. The CoCoViLa tool was successfully used for the development of different DSLs in the domains of simulation of hydraulic systems as well as simulations of security measures for banking and communication networks [7].

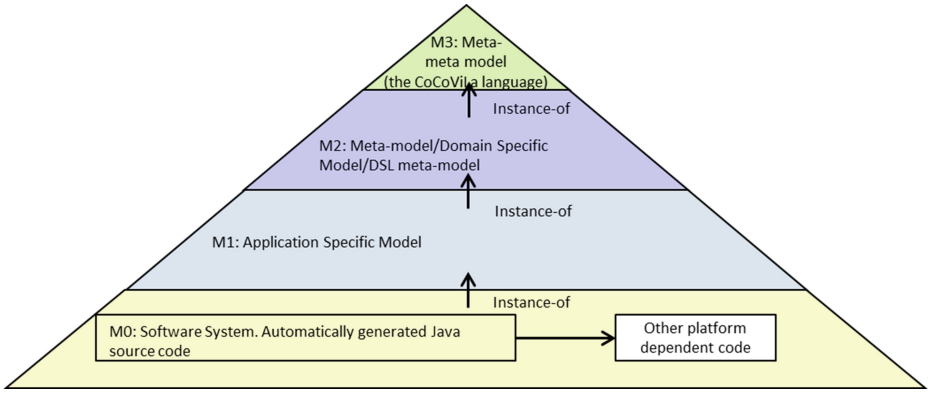


Fig. 1. The meta-pyramid of models based on the CoCoViLa language and tool

In Fig. 1, the current DSL development process with the CoCoViLa system is explained using the OMG MDA terminology [11] and its modelling pyramid. Extensions that constitute the core of this paper are presented in the next sections.

The CoCoViLa modelling language (on the level M3 in Fig. 1) enables to describe meta-models that define DSLs for various domains. Application specific models are created by DSL users. These models of the level M1 are automatically transformed to the corresponding valid logical representation in order to use the method of automatic construction of algorithm of a program [8] and for efficient generation of the corresponding Java source code. The CoCoViLa tool is implemented in Java; therefore the system by default allows generating the Java code directly from an application specific model. Other platform dependent code can be generated from the Java source code.

In Fig. 1, a DSL is created manually taking into account users' requirements and domain knowledge. This corresponds to the previous method of DSL development with CoCoViLa that did not include any special requirements concerning the representation of domain models to be used. In practice, informal methods were used.

Recently, (in [10]) we have presented a method that allows formal domain ontologies presented in OWL to automatically transform to design templates of a DSL meta-model in order to partially automate the DSL development process with CoCoViLa. The current paper takes these transformations into account when extending the original architecture of the CoCoViLa tool.

4 Semantic Integration of Software Artefacts

In order to semantically describe software artefacts used for the development and implementation of a DSL and its meta-model, we use OWL ontologies that are widely utilized in semantic web technological space and supported by well-known standards by W3C. One of the effects of using semantic web standards is an opportunity to semantically describe DSL artefacts and to link software artefacts developed by one modelling tool to artefacts developed by other modelling tools or systems in a distributed way over the Web. Another effect is possibility to use Description Logics (DL) reasoning facilities [1] as ontologies are represented in OWL [9].

4.1 The Concept of DSL Meta-Model Ontology

Central to our approach to integration of software artefacts is the notion of DSL meta-model ontology. We define a *DSL meta-model ontology* as a formal ontology that links together the system ontology and one or more domain ontologies as well as may include links to external software artefacts on the Web (see Fig. 2). In this paper, we consider software artefacts that are needed for the DSL development and application processes. These are for example several types of components of a DSL meta-model like CoCoViLa specifications, Java classes from Java libraries, diagrams and their elements, application packages, the Java source code, etc.

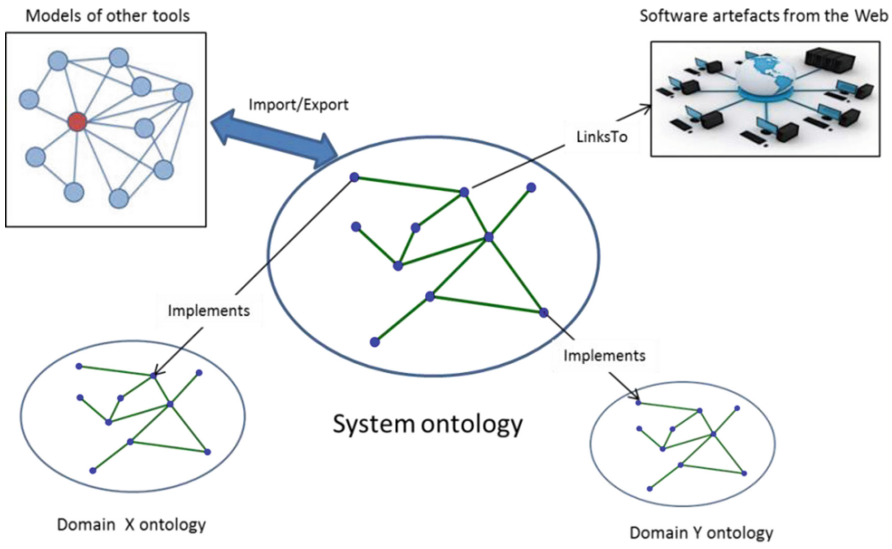


Fig. 2. The concept of DSL meta-model ontology

The *System ontology* formally describes concepts of a particular modelling language and the corresponding software system as well as relationships among them. For example, the CoCoViLa system ontology includes concepts like `JavaClass`, `MetaClass`, `ConceptSpecification`, etc.

The *Domain ontology* provides a specification of a conceptualization of a domain. For example, the geometry domain ontology may contain concepts like `2DimShape`, `Rectangle`, etc.

The DSL meta-model ontology facilitates semantic integration of software artefacts for DSL development and implementation. As shown in Fig. 2, concepts of the system ontology may be related to the concepts of domain ontology via the implementation relationship or they may provide links to external resources used for DSL development. In addition, a part of system ontology concepts may refer to components of models created by the other DSL development tools. This case requires availability of the system ontology of these external tools. In order to ensure the consistency of a DSL meta-model ontology, DL reasoning services are used.

4.2 The CoCoViLa System Ontology

We now explain how the previously described general ontology-based integration approach is implemented for a particular system i.e. the CoCoViLa modelling tool.

The CoCoViLa system ontology formally describes the CoCoViLa modelling language and system concepts (see Fig. 3 for a part of this ontology). The current version of this ontology includes OWL descriptions of 40 classes, 21 object properties and 16 data properties.

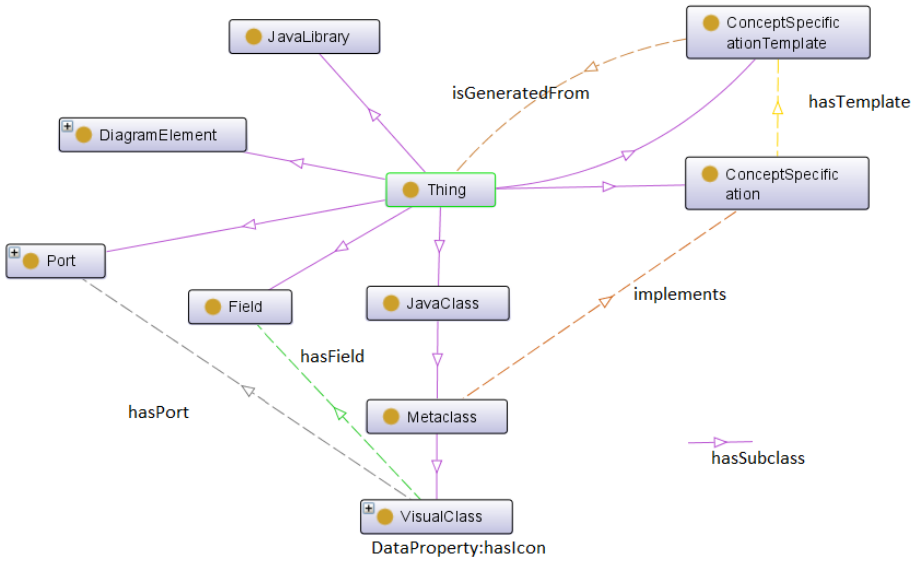


Fig. 3. A fragment of the CoCoViLa system ontology

The `ConceptSpecification` class represents a collection of concept specifications that are textual specifications in the CoCoViLa modelling language. The latter includes definitions of variables, constants, and relations. Concept specifications are used to facilitate automatic composition of an application. Individuals of the

ConceptSpecification class are related to the automatically generated ConceptSpecificationTemplate class instances via the hasTemplate object property.

Concept specification templates are restricted forms of concept specifications that do not include specifications of dynamic parts of a DSL meta-model like relations (except equality relation). These templates are automatically generated from class descriptions of domain ontology using transformation rules given in [10]. The corresponding relationship is represented in ontology by the isGeneratedFrom object property that provides links to domain concepts used in a DSL meta-model. Concept specification templates can be later manually extended with the CoCoViLa language statements that could not be covered by transformations (e.g. equations and relations).

Individuals of the MetaClass class are implemented on the basis of individuals of the ConceptSpecification class and Java. The corresponding relationship is expressed by the implements object property whose domain is the class MetaClass that is a subclass of the class JavaClass collecting instances that are Java classes. The MetaClass class collects individuals that are Java classes and may contain Java methods that are realizations of relations described by the ConceptSpecification class individuals.

It is possible to use diagrammatical elements for a DSL development. Therefore, the CoCoViLa system ontology includes several classes for representing diagram elements as subclasses of the DiagramElement class. Individuals of these elements can come from external sources and be linked via URIs to the DSL meta-model ontology. However, diagrammatical language elements can be also created by the CoCoViLa class editor.

For diagrammatical language of CoCoViLa the notion of a visual class is used. The VisualClass class is a subclass of the MetaClass class. Its individuals are the MetaClass class individuals that are extended with an image, ports and fields. The VisualClass class individuals have the data property hasIcon that could be URI to an image of an icon used for denoting a visual class on a toolbar of the CoCoViLa DSL window.

4.3 An Example: the Geometry DSL Meta-Model Ontology

Let us consider a simple example of the creation of a meta-model ontology for the Geometry DSL. In the following Fig. 4, a fragment of the geometry domain ontology is depicted. All the data property ranges in Fig. 4 are xsd:double.

In order to create a DSL for calculations related to geometric shapes from the given domain ontology of geometric shapes, the CoCoViLa system ontology (see Fig. 3) and domain ontology (see Fig. 4) are imported to the Geometry DSL meta-model ontology. For each (or selected) domain ontology class a corresponding instance of the ConceptSpecificationTemplate class is created and linked to it via the isGeneratedFrom object property value. This object property value indicates from what domain ontology class the template is automatically generated. For example, the following templates in Fig. 5 are generated from the OWL class Rectangle and its super-classes.

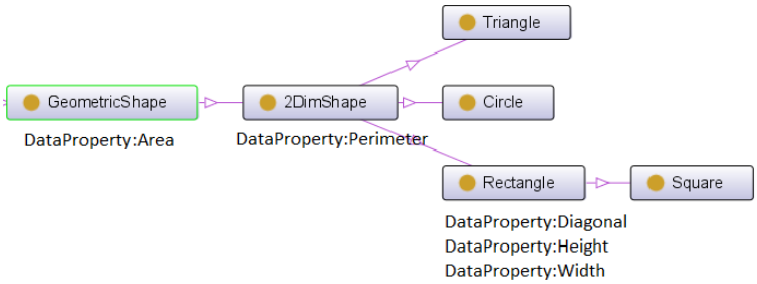


Fig. 4. Domain ontology of geometric shapes (a fragment)

The generated concept specification template of the concept `Rectangle` can be manually completed by corresponding equations for calculating for example a value of the variable `Diagonal`. After that the template becomes the complete concept specification for the concept `Rectangle` that is related to its template via the `hasTemplate` object property value.

```
specification GeometricShape {
            double Area; }

specification _2DimShape super GeometricShape {
            double Perimeter; }

specification Rectangle super _2DimShape {
            double Height, Width, Diagonal; }
```

Fig. 5. A part of a concept specification template in the CoCoViLa textual modelling language.

SubClassOf(sys:MetaClass sys:JavaClass)
SubClassOf(sys:VisualClass sys:MetaClass)
DataPropertyDomain(sys:hasIcon sys:VisualClass)
DataPropertyRange(sys:hasIcon xsd:anyURI)
SubClassOf(geo:Rectangle geo:2DimShape)

Individuals and their relationships

ClassAssertion(sys:ConceptSpecificationTemplate meta:CST1)
ClassAssertion(sys:ConceptSpecification meta:CS1)
ClassAssertion(sys:VisualClass meta:VC1)
ClassAssertion(geo:Rectangle meta:Rectangle)

ObjectPropertyAssertion(sys:isGeneratedFrom meta:CST1 meta:Rectangle)
ObjectPropertyAssertion(sys:hasTemplate meta:CS1 meta:CST1)
ObjectPropertyAssertion(sys:implements meta:VC1 meta:CS1)

DataPropertyAssertion(sys:hasIcon meta:VC1
"http://www.cs.ioc.ee/cocovila/icons/rectangle.png "^^xsd:anyURI)

Fig. 6. The DSL meta-model ontology classes and instances (a fragment)

Some of the Geometry DSL meta-model ontology classes and individuals are represented in Fig. 6. Functional style syntax[5] is used in this figure, where the prefix "sys" denotes the CoCoViLa system ontology elements, the prefix "geo" denotes the geometry ontology elements and the prefix "meta" denotes the Geometry DSL meta-model ontology elements. The Fig. 6 basically shows that the concept specification template `CST1` is generated from the `Rectangle` class of the domain ontology and it is the template for the concept specification `CS1` that is implemented by the visual class `VC1`.

The DSL meta-model makes it possible to link other domains in the analogous way. The consistency of the DSL meta-model ontology is checked by using ontology inference provided by Apache Jena[6].

5 The System Architecture and its Prototypical Implementation

In order to implement a DSL, the original CoCoViLa required a domain expert to transfer the knowledge and a programmer able to convert such informal representation of knowledge into Java classes and annotate these classes with concept specifications. Concept specifications include besides variables also functional dependencies related to concepts. The realizations of functional dependencies can be equations or Java methods implemented in corresponding Java classes. Steps related to the DSL application for solving a particular problem are mostly done automatically by the tool.

The following Fig. 7 depicts the architecture of the CoCoViLa extension that is mainly related to the improvement of a DSL development while components of the previous system (about 80% of the whole system) are used for a DSL application.

The CoCoViLa extension provides facilities for DSL designers to carry out the ontology-based DSL development process that enables the usage of existing formal domain ontologies in combination with the system ontology for a DSL construction.

When loading a DSL, its meta-model ontology (created by DSL designers) is loaded and SPARQL[7] queries are used to dynamically collect and semantically integrate all metadata about artefacts of a DSL meta-model for instantiation of the computational model. Afterwards, the DSL is ready to be used by application developers.

Application developers build the problem specification using the DSL and translate it into the computational model with the help of the CoCoViLa tool. Applying a set of Jena rules enables to extend the computational model with additional relations between concepts in the model. Components (re)used from the previous system are the following: a computational model, the planner, an algorithm and the generated Java code. For more details we refer to [7].

---

5 http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

6 https://jena.apache.org/

7 http://www.w3.org/TR/sparql11-query/

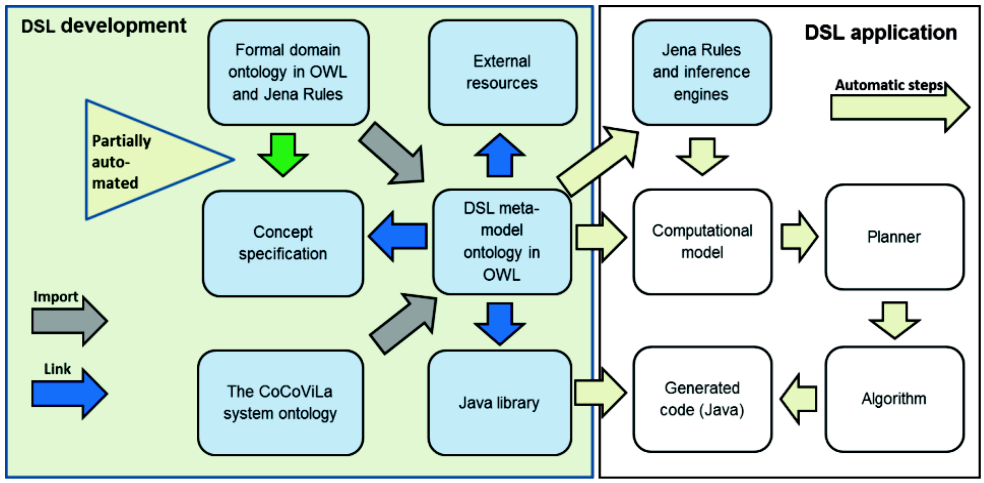


Fig. 7. The CoCoViLa architecture extended with ontology-based DSL development facilities

Computational model is an internal representation of the computational problem and concept specifications. It is used as an input for the planner, a theorem prover, which considers the computational model as a logical theorem with axioms derived from the functional dependencies defined in the specification. Since the prover is based on intuitionistic logic, the solution to the specified computational problem, an algorithm, is extracted from the constructive proof. CoCoViLa generates the Java source code from the algorithm, compiles and executes it at runtime and immediately presents the result of the computation to the user. The generated code can be later (re)used, as it can be saved into the file system.

6 Conclusion

This work demonstrates applicability of formal ontologies for semantic integration of software artefacts for building DSL meta-models. Representing domain models and the system model as OWL ontologies and linking them together to form a unified DSL meta-model ontology makes it possible to effectively integrate software artefacts that constitute a DSL meta-model as well as link it with external resources over the Web. This facilitates dynamic loading (instantiation) of software artefacts of DSL meta-models to a DSL development tool. We have prototypically implemented our approach as an extension to the CoCoViLa DSL modelling tool.

Using the DSL meta-model ontology makes it possible to automatically check its consistency using DL reasoning facilities used for debugging DSL meta-models. It is also easy to incorporate the domain terminology into the DSL at the early development stages due to formalization of domain ontology. Our approach makes it is easy to capture the evolution of a domain in the DSL via automated transformations [10]. However, for semantic integration of artefacts from external tools and models, the approach requires the commitment to a common system ontology or availability of system ontologies of these tools.

Acknowledgements. This research was supported by Estonian Research Council institutional research grant no. IUT33-13, and by the ERDF through the ICT project MBJSDT and Estonian national CoE project EXCS.

References

1. Baader, F., Calvanese, D., McGuiness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
2. Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach. Wiley (2014)
3. Grigorenko, P., Saabas, A., Tyugu, E.: Visual tool for generative programming. In: ESEC/FSE-13: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM, SIGSOFT International Symposium on Foundations of Software Engineering, pp. 249–252. ACM Press, New York (2005)
4. Groenewegen, D.M., Hemel, Z., Kats, L.C., Visser, E.: WebDSL: a domain-specific language for dynamic web applications. In: Harris, G.E. (ed) Proceedings of OOPSLA 2008, pp. 779–780. ACM (2008)
5. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional (2009)
6. Katasanov, A.: Ontology-driven Software Engineering: Beyond Model Checking and Transformations. Int. J. Semantic Computing **06**, 205–242 (2012)
7. Kotkas, V., Ojamaa, A., Grigorenko, P., Maigre, R., Harf, M., Tyugu, E.: CoCoViLa as a multifunctional simulation platform. In: Proc. of the 4th Int. ICST Conference on Simulation Tools and Techniques (SIMUTools 2011), pp. 198–205. ICST, Brussels (2011)
8. Mints, G., Tyugu, E.: Propositional Logic Programming and the Priz System. J. Log. Program **9**(2&3), 179–193 (1990)
9. Motik, B., Patel-Schneider, P.F., Horrocks, I.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. http://www.w3.org/TR/owl2-syntax
10. Ojamaa, A., Haav, H.-M., Penjam, J.: Semi-automated generation of DSL meta models from formal domain ontologies. In: Bellatreche, L., Manolopoulos, Y., Zielinski, B., Liu, R. (eds.) MEDI 2015. LNCS, vol. 9344, pp. 3–15. Springer, Heidelberg (2015)
11. OMG: MDA Guide 1.0.1 (June 2003). http://www.omg.org/mda
12. Puleston, C., Parsia, B., Cunningham, J., Rector, A.L.: Integrating object-oriented and ontological representations: a case study in java and OWL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 130–145. Springer, Heidelberg (2008)
13. Roser, S., Bauer, B.: Automatic generation and evolution of model transformations using ontology engineering space. In: Spaccapietra, S., Pan, J.Z., Thiran, P., Halpin, T., Staab, S., Svatek, V., Shvaiko, P., Roddick, J. (eds.) Journal on Data Semantics XI. LNCS, vol. 5383, pp. 32–64. Springer, Heidelberg (2008)
14. Staab, S., Walter, T., Gröner, G., Parreiras, F.S.: Model driven engineering with ontology technologies. In: Aßmann, U., Bartho, A., Wende, C. (eds.) Reasoning Web. LNCS, vol. 6325, pp. 62–98. Springer, Heidelberg (2010)
15. Walter, T., Parreiras, F.S., Staab, S., Ebert, J.: Joint language and domain engineering. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 321–336. Springer, Heidelberg (2010)
16. Walter, T., Parreiras, F.S., Staab, S.: An ontology-based framework for domain-specific modeling. Software & Systems Modelling **13**, 83–108 (2014)

Publication VII

Haav, Hele-Mai; Ojamaa, Andres (2016).

Semi-Automated Integration of Domain Ontologies to DSL Meta-Models

International Journal of Intelligent Information and Database Systems. [accepted].

The rules of the publisher prevent reprinting of Publication VII. A copy will be provided to opponents and committee members.

Publication VIII

Kivimaa, Jyri; Ojamaa, Andres; Tyugu, Enn (2009).

Graded Security Expert System

Critical Information Infrastructures Security: Third International Workshop, CRITIS 2008, Rome, Italy, October 13–15, 2008, Revised Papers. Ed. Setola, Roberto; Geretshuber, Stefan. Berlin: Springer, 279–286. (Lecture Notes in Computer Science; 5508).

Graded Security Expert System

Jüri Kivimaa[1], Andres Ojamaa[2], and Enn Tyugu[2]

[1] Estonian Defence Forces Training and Development Centre of Communication and Information Systems, Tallinn, Estonia
juri.kivimaa@mil.ee
[2] Institute of Cybernetics at TUT, Tallinn, Estonia
andres.ojamaa@cs.ioc.ee, tyugu@ieee.org

Abstract. A method for modeling graded security is presented and its application in the form of a hybrid expert system is described. The expert system enables a user to select security measures in a rational way based on the Pareto optimality computation using the dynamic programming for finding points of Pareto optimality curve. The expert system provides a rapid and fair security solution for a class of known information systems at a high comfort level.

1 Introduction

Graded security measures have been in use for a long time in the high-risk areas like nuclear waste depositories, radiation control etc. [1]. Also in cyber security, it is reasonable to apply a methodology that enables one to select rational security measures based on graded security, and taking into account the available resources, instead of using only hard security constraints prescribed by standards.

It is well known that complete (100%) security of an information system is impossible to achieve even with high costs. A common practice is to prescribe the security requirements that have to be guaranteed with a sufficiently high degree of confidence for various classes of information systems. This is the approach of most security standards, e.g. [2]. However, a different approach is possible when protecting a critical information infrastructure against the cyber attacks – one may have a goal to provide the best possible defense with given amount of resources (at the same time considering the standard requirements). This approach requires a considerable amount of data that connects security measures with required resources and security measures with provided degree of security.

Practically, only a coarse-grained security can be analyzed in such a way at present, using a finite number of levels (security classes) as security metrics. This is a basis of the graded security methodology. This approach has been successfully applied in the banking security practice and included at least in one security standard [3]. The ideas of graded security are based on the US Department of Energy security model from 1999 [4] and its updated version from 2006 [5].

The graded security model itself is intended for helping to determine a *reasonable* set of needed security measures according to security requirements levels.

R. Setola and S. Geretshuber (Eds.): CRITIS 2008, LNCS 5508, pp. 279–286, 2009.

However, in practice it can be the case that there are not enough resources to achieve the baseline. In this case it is still desirable to invest the limited amount of resources as effectively as possible, i.e. to find and apply an *optimal* set of security measures.

The data required for estimating required resources and security measures can be presented in the form of expert knowledge in an extendable expert system. At present, this expert system can include at least the data that have been used in the banking security design, in particular in a branch of the Swedish bank SEB. Using an expert system has the advantage that it provides flexibility in selecting the required values for the security analysis – the values can be selected based on various input data, and even default values can be used in some non-critical places.

The present paper is organized as follows: the graded security model is presented in Section 2, the optimization method for finding a Pareto optimal curve depending on available resources is described in Section 3, and Section 4 gives a brief overview of the whole software system together with a demo example of security analysis.

2 Graded Security Model

In the present section we briefly explain the basic concepts of the graded security model: security goals, classes and measures as well as costs related to the security measures. We use integrated security metrics for representing the overall security of a system. We explain the way these entities are related.

Conventional goals of security are confidentiality, integrity, availability, and non-repudiation. In this presentation, that is based mainly on banking security, we use the following four slightly different *security goals: confidentiality (C), integrity (I), availability (A)* and *satisfying mission criticality (M)*. (The latter two are in essence two aspects of availability.) The model can be extended by including additional security goals. A finite number of levels are introduced for each goal. At present, we use four levels *0, 1, 2, 3* for representing required security, but the number of levels can vary for different measures. The lowest level 0 denotes absence of requirements.

Security class of a system is determined by *security requirements* that have to be satisfied. It is determined by assigning levels to goals, and is denoted by respective tuple of pairs, e.g. C2I1A1M2 for the system that has second level of confidentiality C, first level of integrity I etc.

To achieve the security goals, some security measures have to be taken. There may be a large number of measures. It is reasonable to group them into *security measures groups*. Let us use the following nine groups in our simplified examples which are based on an educational information assurance video game CyberProtect [6]: user training (1), antivirus software (2), segmentation (3), redundancy (4), backup (5), firewall (6), access control (7), intrusion detection (8), and encryption (9).

The number of possible combinations of security levels for all security goals is $4^4 = 256$. This is the number of different security classes in our case, see

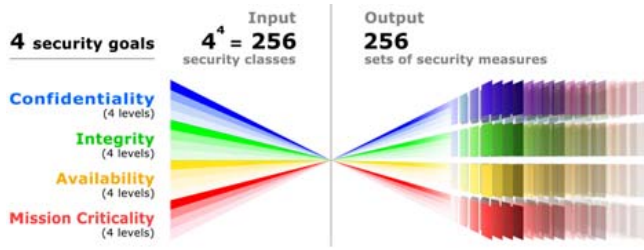


Fig. 1. Security classes of graded security model

Fig. 1. A security class determines required security levels for each group of security measures. *Abstract security profile* is an assignment of security levels (0, 1, 2 or 3) to each group of security measures. Hence, in the present example, we have totally $4^9 = 2621144$ abstract security profiles to be considered. The number of security measures groups may be larger in practice, e.g. 20. This gives a big number of abstract security profiles – 4^{20} for 20 groups. Knowing the costs required for implementing security measures of any possible level, one can calculate the costs of implementing a given abstract security profile.

After selecting security levels for a security measures group, one can find a set of concrete measures to be taken. This information is kept in the knowledge modules of the expert system of security measures, see Section 4.

It is assumed that, applying security measures, one achieves security goals with some confidence. The security confidence l_i is described by a numeric value between 0 and 100 for each group of security measures $i = 1, \ldots, n$, where n is the number of groups.

We describe overall security of a system by means of an integrated security metrics – the security is evaluated by *weighted mean security confidence* S:

$$S = \sum_{i=1}^{n} a_i l_i \ ,$$

where l_i is security confidence of i-th security measures group, a_i is a weight of the i-th group, $i = 1, \ldots, n$, and

$$\sum_{i=1}^{n} a_i = 1 \ .$$

Information about costs, required security measures and confidence levels needed for calculations is presented in the expert system that will be described in Section 4.

Graded security methodology as it is generally accepted, enables one to find required security measures and costs for a given security class. We add also the value of weighted mean security confidence S. Fig. 2(a) shows a usual graded security solution: the value of S for given resources r, and also selected security levels of security measures groups. The levels for the groups numbered from 1 to 9 are shown on the right side scale.

3 Optimization Technique

Our expert system allows us to solve several security related optimization problems. First of all, it enables one to find an optimal security solution for given resources, and to determine the reachable security class. This problem concerns again only one value of resources, and can be illustrated by the same picture as the conventional graded security problem (Fig. 2(a)).

To get a broader view of possible solutions, one should look at the optimal security for many different values of usable resources. This service is provided by our expert system by plotting a Pareto optimality tradeoff curve that binds resources and the achievable security S. Fig. 2(b) shows this curve for an interval of resources from r_1 to r_2. The last value of resources r_2 can be easily calculated as the resources required for getting the security class `C4I4A4M4`. The curve shows also the respective security levels for selected security measures groups – in the present case, for the groups number 1 and 4. The exhaustive search of optimal solutions for q possible values of resources, n security measures groups and k security levels requires testing (calculating weighted mean confidence) of qk^n points.

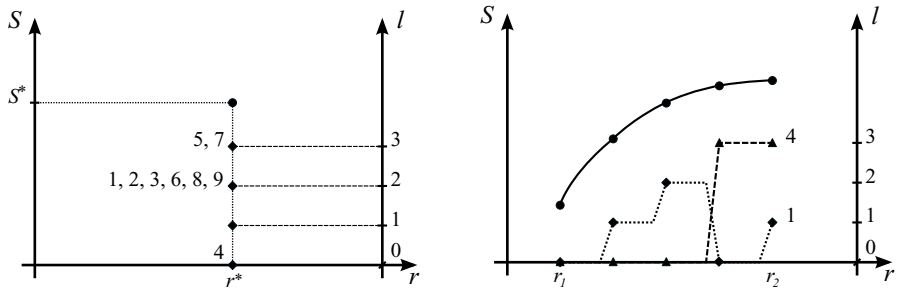
Building optimal solutions gradually, for $1, 2, \ldots, n$ security measures groups enables us to use discrete dynamic programming, and to reduce considerably the search. Indeed, the fitness function S defined on intervals from i to j as

$$S(i,j) = \sum_{s=i}^{j} a_s l_s$$

is additive on the intervals, because from the definition of the function S we have

$$S(1,n) = S(1,s) + S(s,n), 1 < s < n \ .$$

This means that one can build an optimal resource assignment to security measures groups gradually, as a path in the space with coordinates x_1, x_2, where x_1 equals to the number of security measures groups that have got resource (i.e.



(a) Conventional graded security solution

(b) Pareto-optimal solutions

Fig. 2. Conventional graded security solution and Pareto optimality tradeoff curve

$x_1 = s$) and x_2 equals to the amount of used units of resources. This algorithm requires testing of q^2nk points (q is number of possible values of resources, n is number of security measures groups and k is number of security levels).

4 Security Expert System

A hybrid expert system with visual specification language for security system description has been built on the basis of a visual programming environment CoCoViLa [7]. The system includes knowledge modules (rule sets) in the form of decision tables for handling expert knowledge of costs and gains, as well as for selecting security measures for each security group depending on the required security level. Other components are an optimization program for calculation Pareto optimality curve parameterized by available resources, and a visual user interface for graphical specification of the secured system, visual control of the solution process through a GUI, and visualization of the results. These components are connected through a visual composer that builds a Java program for each optimization problem, compiles and runs it on the request of the user, see Fig. 3.

Let us explain the usage of the expert system on the following simplified example. We have nine security measures groups as given in Section 2. Two groups – "user training" and "encryption" – have specific values of cost and confidence related to security levels that must be given as an input. We can use standard values of cost and confidence given in the expert knowledge modules for other groups. We have to solve the problem in the context of banking and can use resources measured in some units on the interval from 1 to 70. The security class `C2I1A1M2` is given as an input. The expected outcome is a graph that shows the weighted mean security confidence depending on the resources that are used in the best possible way. The graph should also indicate whether the security goals specified by the security class can be achieved with the given amount of resources. Besides that, the curves showing security confidence provided by user training and redundancy must be shown.

The visual composer is provided by the CoCoViLa system that supports visual model-based software composition. The main window of the expert system shown in Fig. 4 presents a complete description of the given problem. It includes also visual images of components of the expert system and a toolbar for

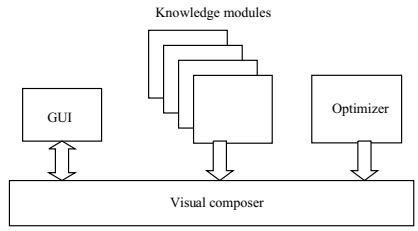


Fig. 3. Graded security expert system

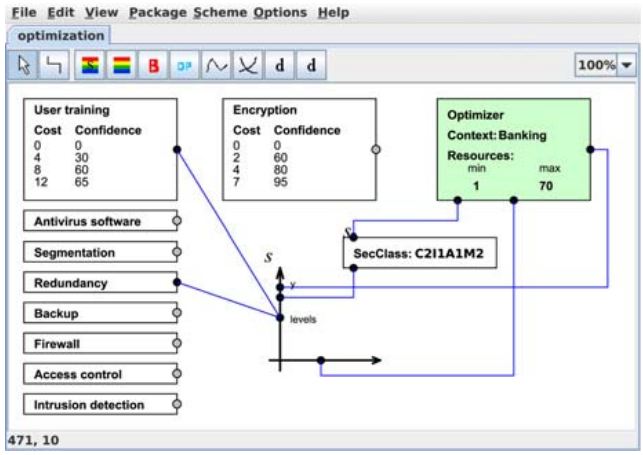


Fig. 4. Problem specification window

adding new components, if needed. In particular, new security measures groups can be added by using the third and fourth button of the toolbar. Besides the security measures groups there are three components – Optimizer, SecClass and GraphVisualizer – shown in the window. The components in the main window can be explicitly connected through ports. This allows us to show which values of security should be visualized ("user training" and "redundancy" in the present case) etc. There are extended views of two security measures groups – "user training" and "encryption" that have explicit values of costs and confidence given as an input. Other groups use the standard values of costs and confidence given in the expert knowledge modules as specified in the problem description. The SecClass component is used for specifying security goals. During computations, this component also evaluates the abstract security profiles calculated by the Optimizer against the actual security requirements using a knowledge module from the expert system.

5 Optimization Results

As an example, in Fig. 5 there is a window showing the optimization results. The upper curve (Confidence) represents the optimal value of weighted mean security confidence depending on the resources that are used in the best possible way. This curve is further divided into four parts to visualize to which degree the optimal result satisfies the security requirements given by the security class. The first part (black line) indicates the interval of resources where none of the four (in our example) security goals can be achieved. The second part (grey line, three separate segments) shows that at least one of the security goals is satisfied while also at least one is not. The third part (thick black line) represents the amount of resources that, when used optimally, would result in satisfying the requirements exactly. One should note that this coincidence of the optimal security profile and

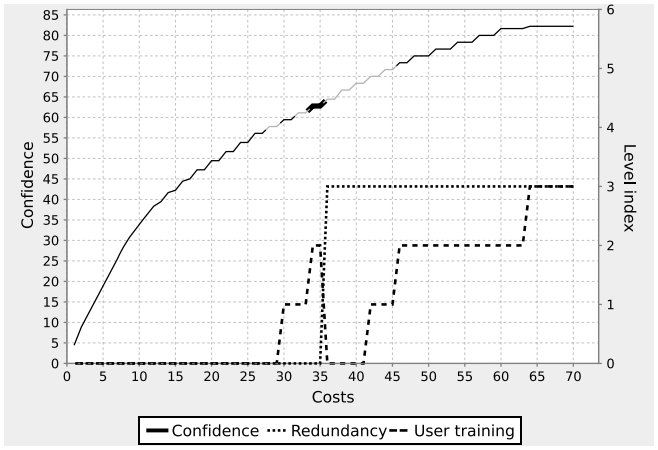


Fig. 5. Solutions window

the security requirements does not always exist. The last part of the graph (black line, again) shows the amounts of resources that are more than is strictly needed to satisfy the requirements. It is interesting to notice that on the interval of costs from 36 to 45 units it is possible to satisfy all security goals, because already spending 34 units enables one to do this. However, the solutions with highest values of the weighted mean security confidence do not satisfy all security goals on this interval.

The lower graphs indicate (on the right scale) the optimal levels of two measures groups corresponding to the given amount of resources. These graphs are not necessarily monotonic as can be seen in this example at the resource values 35 and 36. When there are 35 units of resources available it is reasonable to apply the measure "user training" at level 2. Having one more unit of resources better overall security confidence level is achieved by taking all resources away from "user training" and investing into the "redundancy" measures group to achieve level 3.

6 Conclusions

The advantage of the expert system of the graded security is that it provides a rapid security solution at a sufficiently high, although not 100%, confidence level. Based on our previous experience, the graded security expert system allows a typical security solution to be developed within approximately 8 hours, with about half the time spent on security class identification and the other half on manually analyzing available resources, accepted security risks, attack costs and other optimization variables. Our method reduces the time for analysis to a few seconds by automatic optimization and presenting a global view in the form of a Pareto optimal solution. It includes:

- graded security selection procedure that yields the security measures for a given security class;
- high-level analysis of usage of resources for information security and accepted risks based on advanced optimization technique.

We understand that wider application of this method will depend on the availability of expert knowledge that binds costs and security confidence values with taken security measures. This knowledge can be collected only gradually, and will depend on the type of the critical infrastructure that must be protected.

Acknowledgements. We thank the Estonian Ministry of Defence and the Estonian Defence Forces Training and Development Centre of Communication and Information Systems for the support of this work. The contribution of the second author was partially supported by the Estonian Information Technology Foundation and the Tiger University program.

References

1. Kang, Y., Jeong, C. H., Kim, D. I.: Regulatory approach on digital security of instrumentation, control and information systems in nuclear power plants. Korea Institute of Nuclear Safety. Daejeon, Korea, `http://entrac.iaea.org/IandC/TM_IDAHO_2006/CD/IAEA%20Day%202/Kang%20paper.pdf` (August 31, 2008)
2. German Federal Office for Information Security (BSI): IT Baseline Protection Manual (2005), `http://www.bsi.de/gshb/` (August 31, 2008)
3. Estonian Information Systems Three-Level Security Baseline System – ISKE ver. 1.0
4. U. S. Department of Energy, Office of Security Affairs: Classified Information Systems Security Manual (1999)
5. U. S. Department of Defense: National Industrial Security Program Operating Manual (NISPOM) (2006)
6. U. S. Department of Defense, Defense Information Systems Agency. CyberProtect, version 1.1 (July 1999), `http://iase.disa.mil/eta/product_description.pdf` (August 31, 2008)
7. Grigorenko, P., Saabas, A., Tyugu, E.: Visual tool for generative programming. ACM SIGSOFT Software Engineering Notes 30(5), 249–252 (2005)

Publication IX

Ojamaa, Andres; Tyugu, Enn; Kivimaa, Jyri (2008).

Pareto-Optimal Situation Analysis for Selection of Security Measures

MILCOM 08: Assuring Mission Success: Unclassified Proceedings, November 17–19 San Diego. 3224–3230.

PARETO-OPTIMAL SITUATON ANALYSIS FOR SELECTION OF SECURITY MEASURES

Andres Ojamaa and Enn Tyugu
Institute of Cybernetics of
Tallinn University of Technology
Tallinn, Estonia

Jyri Kivimaa
Estonian Defence Forces Training and Development
Centre of Communication and Information Systems
Tallinn, Estonia

ABSTRACT

A methodology of selection of security measures is presented and a prototype implementation in the form of a hybrid expert system is described. This expert system is applicable, first of all, in the security management. It enables a user to select security measures in a rational way based on the Pareto optimality computation using a discrete dynamic programming method. This enables one to select rational countermeasures taking into account the available resources instead of using only hard constraints prescribed by standards. The prototype expert system is presented that provides a rapid security solution for a class of known information systems. Coarse-grained security can be analyzed in such a way at present, using a finite number of levels (security classes) as security metrics. This is a basis of the graded security methodology.

1. INTRODUCTION

Selection of security measures is a complex problem due to the fact that multiple objectives must be achieved at the same time. Considering data security, the security goals can be confidentiality, integrity and availability. Besides that, a security officer may want to keep costs reasonably low from one side, and reach the security goals with as high confidence as possible. Low cost and high confidence are two universal goals. The complexity has been an obstacle to finding optimal solutions for the security management problem. Another obstacle has been the absence of reliable metrics for measuring the said goals[1].

[1] "Good metrics are those that are SMART, i.e. specific, measurable, attainable, repeatable, and time-dependent, according to George Jelen of the International Systems Security Engineering Association [1]. Truly useful metrics indicate the degree to which security goals, such as data confidentiality, are being met, and they drive actions taken to improve an organization's overall security program [2]."

978-1-4244-2677-5/08/$25.00

Graded approach has been applied earlier in standards covering areas other than information security [3]. In recent years a graded security method has been developed and used in a number of areas, not necessarily in information assurance [4]. This method relies on a coarse-grained metrics for the security goals and achieved confidences. It is successfully applied as a basis for security standards that prescribe concrete security measures for achieving a required level of confidence for each security goal [5, 6]. The method is not immediately applicable for finding an optimal solution of the security problem.

We are going here to use the metrics of the graded security method and build a model that binds taken security measures with costs and confidences of achieving the goals. We introduce a fitness function that presents by one numeric value the integral confidence of achieving the security goals. This allows us to formulate a problem of selecting security measures as an optimization problem in precise terms. However, we still have two goals: to minimize the costs and to maximize the integral security confidence. This problem will be solved by means of building a Pareto optimality tradeoff curve that explicitly shows the relation between used resources and security confidence. Then, knowing the available resources, one can find the best possible security level that can be achieved with the resources and find the security measures to be taken. From the other side – if the required security level is given one can find the resources needed and the measures that have to be taken. This requires solving an optimization problem for each value of resources. As the number of possible security measures (that are in principle the independent variables of the optimization problem) is large, we have grouped the measures into security measures groups that will be characterized by security confidence levels. Taking the confidence levels of the groups as independent variables, we get an optimization problem of a reasonable size that can be solved by means of a discrete dynamic programming method.

The presented method of finding optimal security measures is in principle applicable in different situations, in particular, for designing overall security of a communication network, for designing a security of a critical information infrastructure of a bank etc. However, the method requires considerable amount of data that bind costs and confidences with security measures groups as well as expert knowledge that binds concrete security measures with a selected security confidence requirements level of a group. In the end of the present paper we give an example of an expert system developed for banking security that has the data and has been used for experimenting. Most of the expert knowledge of this kind can be extracted from standards or internal security policies of the bank or other organization that must have them before trying to optimize the security.

2. GRADED SECURITY MODEL

In the present section we briefly explain the basic concepts of the graded security model that gives functional dependencies for our optimization method. We are going to use integrated security metrics for representing the overall security of a system. Conventional goals of security are *confidentiality (C), integrity (I), availability (A),* and *non-repudiation (N)*. The model can be extended by including additional security goals. A finite number of security levels are introduced for each goal. This is a coarse-grained metrics, but the only available in this context at present. We use four levels 0, 1, 2, 3 for representing required security, but the number of levels can vary for different measures. The lowest level 0 denotes absence of special protective measures. *Security class* of a system is determined by security requirements that have to be satisfied. It is determined by assigning levels to goals, and is denoted by a respective tuple of pairs, e.g. `C2I1A1N2` for the system that has second level of confidentiality `C`, first level of integrity `I` and availability `A` and second level of non-repudiation `N`.

To achieve the security goals, proper *security measures* have to be taken. There may be a large number (hundreds) of measures. It is reasonable to group them into *security measures groups* g_1, g_2, …, g_n. The grouping should be done in such a way that measures of one and the same group will be always used for achieving one and the same level of security. We will need a function f that produces a set of required security measures $f(l, g)$ for a given security measures group g and a security level l of the group.

A security class determines the required security level for each group of security measures. Let us denote by s a respective function that produces a security level $s(c, g)$ for a group g when the security class is c. *Abstract security profile* is an assignment of security levels (0, 1, 2 or 3) to each group of security measures. This can be expressed by the tuple $p = (s(c, g_1), s(c, g_2), \ldots, s(c, g_n))$, where p denotes the abstract security profile an the elements of the tuple p are indexed and appear in the tuple in the same order as the groups of security measures $g_1, g_2, \ldots, g_n$ have been indexed.

For n security measures groups we have totally 4^n abstract security profiles to be considered. The number of security measures groups may be in practice up to 20 or even more. This gives a number of abstract security profiles: 4^{20}. (If we had considered all security measures without grouping them, then we had got an incomprehensibly large number of security profiles – 4^k, where k is several hundreds.)

Knowing the cost function h that gives the costs $h(l, g)$ required for implementing security measures of a group g for a level l, one can calculate the costs of implementing a given abstract security profile:

$$costs(p) = \sum_{i=1}^{n} h(l_i, g_i), \text{ where } p = (l_1, l_2, \ldots, l_n)\,.$$

Our goal is to keep the value $costs(p)$ as low as possible.

The information for calculating values of functions f, h, c and s should be kept in the knowledge modules of an expert system of security measures.

It is assumed that, applying security measures, one achieves security goals with some confidence. The security confidence q of a group g that satisfies the security level l is given by a function $q(l, g)$ and it is a numeric value between 0 and 100 for each group of security measures.

We describe overall security of a system by means of an *integrated security metrics* that is a weighted mean security confidence S, called also *integral security confidence*:

$$S = \sum_{i=1}^{n} a_i q_i\,,$$

where q_i is security confidence of the i-th security

measures group, a_i is a weight of the i-th group, and

$$\sum_{i=1}^{n} a_i = 1\,.$$

In the simplest case $a_i = 1/n$, and the integral security confidence is the average confidence of security measures groups. Also the information about the weights a_i, as well as about the function q must be presented in an expert system.

3. OPTIMIZATION TECHNIQUE

Now we can formulate an optimization problem as follows: "find the abstract security profile p with the best (highest) value of integral security confidence S for given amount of resources r, so that $costs(p) \leq r$." We have introduced all functions needed for calculating S and $costs$ in the previous section. Independent variables whose values have to be found by optimization are the security levels assigned to security measures groups: $l_1, l_2, \ldots, l_n$. If the security class c is given, then the solution has to satisfy also the constraints

$$l_i \geq s(c_i, g_i),\ \ i = 1, 2, \ldots, n\,.$$

Remark. The graded security model presented in Section 2 is usually used for finding (for a given security class) the required security levels of security measures groups and respective costs and concrete measures to be taken. This problem is considerably simpler that the optimization problem considered here.

Let us solve the optimization problem in the general case when also a security class is given. First, a security class prescribes only minimal security requirements and respectively – spending of some minimal amount of resources r_{min}. It is easy to calculate also resources r_{max} that can be reasonably spent for achieving the maximal possible integrated security level –

$$S_{max} = \sum_{i=1}^{n} a_i q_{max\,i}\ ,$$

where $q_{max\,i}$ is maximal security confidence of the i-th group of security measures.

Applying some resources between the values r_{min} and r_{max}, one can get better security in a rational way. Now we have an optimization problem with two goals: to minimize resources on the interval $[r_{min}, r_{max}]$ and to maximize security, guaranteeing at least the levels prescribed by a given security class. We reduce this problem to a simpler one that will be solved many times. We find an optimal security solution – the solution that has maximal value of S, for a fixed value of resources. We repeat this optimization for as many values of resources as needed. In this way we get a curve that shows the best possible value of fitness function S for every value of resources used, see Fig. 1. This curve is called a *Pareto optimality tradeoff curve* for resources and security. In the case when the minimal security requirements are not strict for security measures groups, it is reasonable to compute Pareto optimality even for resources less than r_{min}. This can be done, if the optimization procedure is sufficiently fast, like in our case.

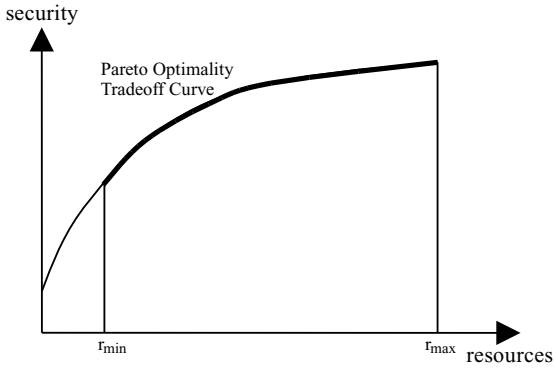


Figure 1. Search of optimal security along resource dimension

The exhaustive search of optimal solutions for m possible values of resources, n security measures groups and k security levels requires testing (calculating weighted mean confidentiality) of mk^n points.

Building optimal solutions gradually, for $1, 2, \ldots, n$ security measures groups enables us to use discrete dynamic programming, and to reduce considerably the search. Indeed, the fitness function S defined on intervals from j to k as

$$S(j, k) = \sum_{i=j}^{k} a_i l_i\,,$$

is additive on the intervals, because from the definition of the function S we have

$$S(1, n) = S(1, k) + S(k, n)\,.$$

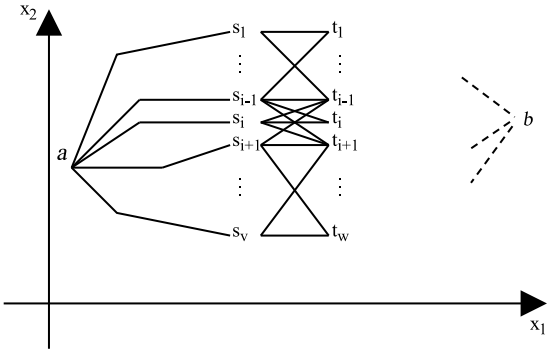


Figure 2. Resource assignment by means of discrete dynamic programming

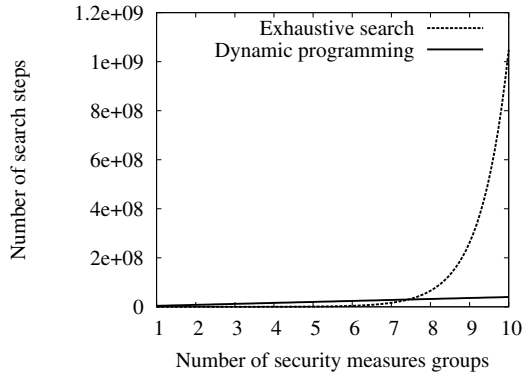


Figure 3. Complexity of search

This means that one can build an optimal resource assignment to security measures groups gradually, as a path in the space with coordinates x_1, x_2, where x_1 equals to the number of security measures groups that have got resource (i.e. $x_1 = k$) and x_2 equals to the amount of used units of resources ($1, 2, \ldots, 1000$ in our example). The discrete dynamic programming method requires using of a finite number of values of a resource (x_2). This number of values depends on the precision that is required. A precision that can be achieved using expert knowledge is not very high, usually a hundred points is sufficient. As our optimization procedure works sufficiently fast we are using 1000 points. Fig. 2 shows a search step, where known optimal partial solutions (assignments of resources to already tested security measures groups) are the paths from initial state a (where no resources are assigned) to intermediate states $s_1, \ldots, s_v$. The aim is to find one step longer optimal paths from a to the states $t_1, \ldots, t_w$ that follow the states $s_1, \ldots, s_v$. This can be done for each step by trying out all possible continuations of the given partial optimal paths to $s_1, \ldots, s_v$ as shown in Fig. 2. This algorithm requires testing of $m^2 n$ points (m is number of possible values of resources, n is number of security measures groups).

In Fig. 3 it is shown how the number of search steps (and consequently search time) depends on the number of security measures groups for the number of groups up to 10. Our method has linear complexity, the search time grows linearly with the number of groups. The time for exhaustive search grows exponentially.

4. APPLICATION EXAMPLE

We have developed a prototype of a security expert system for selecting security measures in banking. This expert system has been developed in a visual programming environment CoCoViLa [8]. Let us explain its functioning on an example. Here we use the following four security goals: confidentiality (C), integrity (I), availability (A) and satisfying mission criticality (M). We use the following nine security measures groups in our simplified example which are based on an educational information assurance video game CyberProtect [7]:

- firewall,
- access control,
- intrusion detection,
- encryption,
- user training,
- antivirus software,
- segmentation,
- redundancy,
- backup.

After selecting security levels for a security measures group, one can find a set of concrete measures to be taken. For example, in the case of the security level 1 for the group "User training" the following measures can be found from the expert system:

- New employees must be instructed for security – procedures and practice must be explained.
- An employee must know security related rights and obligations, must understand security practice, know about handling of passwords and keys.

- An employee must be instructed about security regulations and should be motivated to follow the regulations. Help about security must be available for all IS users.

The main window of the expert system is presented in Fig. 4. A menu bar in the upper part of the window provides buttons for building a visual specification of the problem to be solved. These are buttons for adding different kinds of components, and tools for manipulating components on the scheme and connecting components through ports. From left to right the buttons are:

- selection tool,
- connection line,
- simple security measures group (uses default values defined by its name and current context),
- expanded security measures group with explicit values,
- brute-force optimizer,
- dynamic programming optimizer,
- simple graph,
- multiple graphs,
- security class evaluator,
- extended security class evaluator.

The window includes a visual specification of our problem. The specification is a scheme where components are security measures groups and other software components that are used for solving the problem. The scheme has been composed gradually by adding and connecting new components and editing properties of the components. In the given scheme we see images of all security measures groups. Besides the security measures groups there are three components *Optimizer*, *SecClass* and *GraphVisualizer* shown in the scheme. Two groups "User training" and "Encryption" are represented by expanded components that have specific values of cost and confidence related to security levels explicitly given as an input. Standard values of cost and confidence are used for other groups. They are given in the expert knowledge modules. We have to solve the problem in the context of banking and can use resources measured in some units on the interval from 1 to 70 that is shown in the *Optimizer* block. The security class `C2I1A1M2` is given as a separate block as well. Some blocks in the main window are connected through ports. This allows us to show which values of security should be visualized ("User training" and "Redundancy" in the present case) etc. The expected outcome is a graph produced by the *GraphVisualizer* that shows the weighted mean security confidence depending on the resources that are used in the best possible way. The graph should also indicate whether the security goals specified by the security class can be achieved with the given amount of resources. Besides that, the curves showing security confidence provided by user training and redundancy will be shown, see the respective connection lines between the visual images.

5. OPTIMIZATION RESULTS

In Fig. 5 there is a window showing the optimization results. The upper curve (Confidence) represents the optimal value of weighted mean security confidence depending on the resources that are used in the best possible way. This curve is further divided into four parts to visualize to which degree the optimal result satisfies the security requirements given by the security class. The first part (thin black line for the costs up to 27) indicates the interval of resources where none of the four (in our example) security goals can be achieved. The second part (thin grey line, three separate segments) shows that at least one of the security goals is satisfied while also at least one is not. The third part (thick black line) represents the amount of resources that, when used optimally, would result in satisfying the requirements exactly. One should note that this coincidence of the optimal security profile and the security requirements does not always exist. The last part of the graph (thin black line, again) shows the amounts of resources that are more than is strictly needed to satisfy the requirements. It is interesting to notice that on the interval of costs from 36 to 45 units it is possible to satisfy all security goals, because already spending 34 units enables one to do this. However, the solutions with highest values of the weighted mean security confidence do not satisfy all security goals on this interval.

The lower graphs indicate (on the scale shown on the right) the optimal levels of two measures groups corresponding to the given amount of resources. These graphs are not necessarily monotonic as can be seen in this example at the resource values 35 and 36. When there are 35 units of resources available it is reasonable to apply the measure "User training" at level 2. Having one more unit of resources, a better overall security confidence level is achieved by taking all resources away from "User training" and investing into the "Redundancy" measures group to achieve level 3.

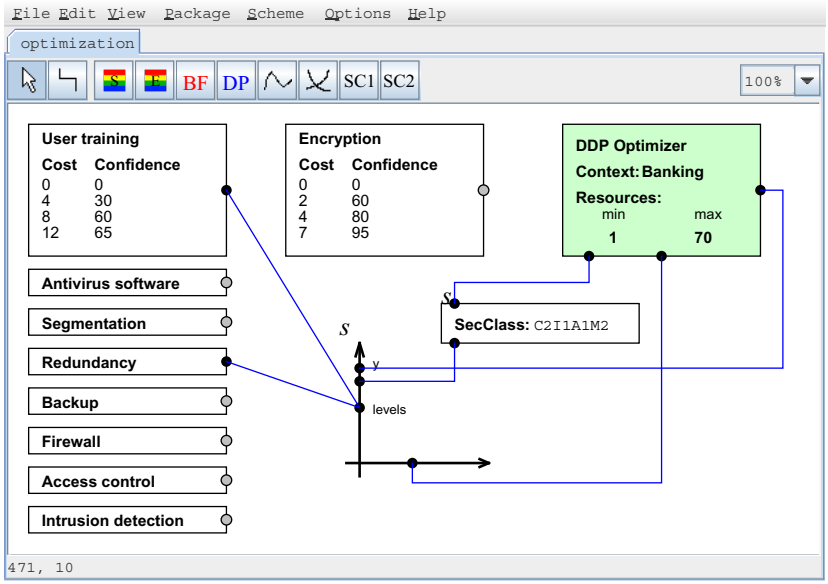


Figure 4. Main window of security expert system

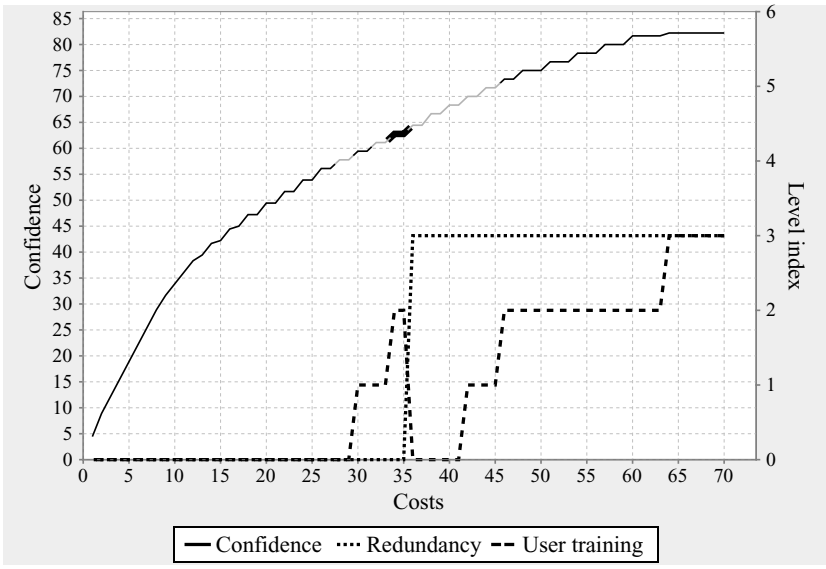


Figure 5. Solution of the problem

6. CONCLUDING REMARKS

In the present work we have developed a method for systematic design of a security solution of an information or communication system, and the method is explained on an example from the banking security. The method relies on a graded security model used in practice in different applications. The novelty of the method is, first, the usage of an advanced optimization technique based on discrete dynamic programming and, second, the output of many alternative solutions in the form of a Pareto optimality tradeoff curve that enables the user to select the best security solution depending on availability of resources.

Another novelty is introduction and usage of an integral security measure in the form of a weighted mean security confidence. The method performs security situation analysis using coarse-grained metrics for security levels of partial solutions (security measures groups) from one side, and an integrated security metrics in the form of weighted mean security confidence from the other side. A tool developed as a prototype supports visual presentation of a general view of a security situation and enables one to perform the situation analysis on different levels of details, e.g. using standard functions of confidences and costs or presenting them as additional inputs. Time required for automated analysis, when a set of input data is given, is only a few seconds. This enables one to perform the analysis rapidly for many different assumptions.

We understand that wider application of this method will depend on the availability of expert knowledge that binds costs and security confidence values with taken security measures. This knowledge can be collected only gradually, and will depend on the type of the critical infrastructure that must be protected. However, our expectation is that more expert knowledge will be collected when interactive analysis applications with graphical user interface such as the prototype presented in this paper become available.

ACKNOWLEDGEMENTS

We thank the Estonian Ministry of Defence and the Estonian Defence Forces Training and Development Centre of Communication and Information Systems for the support of this work. The first author would like to thank the Estonian Information Technology Foundation and the Tiger University program for partial support of this work.

REFERENCES

[1] G. Jelen. *SSE-CMM Security Metrics*. NIST and CSSPAB Workshop, Washington, D.C., 13–14 June 2000.

[2] S. C. Payne. *A Guide to Security Metrics*. SANS Reading Room, 2006. http://www.sans.org/reading_room/whitepapers/ (10 Sep 2008)

[3] C. E. Pasterczyk. *A graded approach to ISO 9000 implementation for records managers*. Association of Records Managers and Administrators international annual conference, Toronto (Canada), 25–29 September 1994.

[4] Y. Kang, C. H. Jeong, D. I. Kim. *Regulatory approach on digital security of instrumentation, control and information systems in nuclear power plants*. Korea Institute of Nuclear Safety. Daejeon, Korea. http://entrac.iaea.org/I-and-C/TM_IDAHO_2006/CD/IAEA%20Day%202/Kang%20paper.pdf (10 Sep 2008)

[5] German Federal Office for Information Security (BSI). *IT Baseline Protection Manual*. 2005. http://www.bsi.de/gshb/ (10 Sep 2008)

[6] U. S. Department of Defense. *National Industrial Security Program Operating Manual (NISPOM)*. 2006.

[7] U. S. Department of Defense, Defense Information Systems Agency. *CyberProtect, version 1.1*. July 1999. http://iase.disa.mil/eta/product_description.pdf (10 Sep 2008)

[8] P. Grigorenko, A. Saabas, E. Tyugu. *Visual tool for generative programming*. ACM SIGSOFT Software Engineering Notes, 2005, 30, 5, 249–252.

Publication X

Kivimaa, Jyri; Ojamaa, Andres; Tyugu, Enn (2009).

Managing Evolving Security Situations

MILCOM 2009: Unclassified Proceedings, October 18–21, 2009, Boston, MA. Piscataway, NJ: IEEE, 1–7.

Paper ID# 900425.PDF

MANAGING EVOLVING SECURITY SITUATIONS

Jyri Kivimaa
Cooperative Cyber Defence
Centre of Excellence
Tallinn, Estonia

Andres Ojamaa
Institute of Cybernetics at
Tallinn University of Technology
Tallinn, Estonia

Enn Tyugu
Cooperative Cyber Defence
Centre of Excellence
Tallinn, Estonia

ABSTRACT

A method is described that takes into account the investments done in the security and/or achieved security confidence in planning new security measures. The method uses new integral security metrics and the well-known graded security model. A precondition for the application of this method is the availability of expert knowledge or statistical data for the model in use that describes a class of situations where the analyzed security situation belongs to. For a number of situations at present, this information has been extracted from standards of graded security. For specific military communications applications the data must be collected from a log analysis of characteristic attacks and security reports, as well as by the traditional knowledge acquisition means.

1. INTRODUCTION

The security situation in cyber space is changing rapidly. This requires continuous analysis of security situations and continuous security management: selection of security measures, planning of investments for security measures groups. Our goal is to provide a method for planning security measures not only for a fixed time point, but to do this for a longer time period, possibly, investing into the security gradually. This paper presents a method that is an extension of the Pareto-optimal security situation analysis implemented in an expert system [4]. It takes into account the legacy systems and security levels achieved by means of former investments. This enables one to plan the usage of resources considering evolving security situations over a longer time period.

Comprehensive security planning is a complex task. This can be seen from the complexity of standards and requirements like Common Criteria [7] or ISKE [1]. Standards prescribe minimal required measures, and usually do not include economic parameters—the costs of implementing the security measures. A detailed cost-benefit analysis of cyber security [2] may require months. An alternative approach is to manage security on the basis of security requirements. It is efficient, if reasonably good expert knowledge of security requirements and goals is available. We have taken this approach.

A well-known graded security methodology [6, 8] is based on a comprehensive but coarse grained model, and provides a way of planning security and calculating costs. In our paper [4] we have shown how to use the graded security model for finding optimal solutions depending on the given security situation. However, a description of a situation there reflects neither the investments already done into security nor the levels of security already achieved. Based on the application of a discrete dynamic programming method described in [5], one can solve rather complex security optimization problems on ordinary PCs and laptops. This enabled us to extend the optimization method for longer time intervals, solving the optimization problem stepwise.

This paper is organized as follows. In the next section we present briefly the graded security method that provides the functional dependencies needed for calculations. A separate section (Section 3) is devoted to the discussion of the integral security metrics needed for comparing the solutions. These metrics were introduced for the first time in [4]. The following Section 4 includes a brief description of the software used for making calculations. Section 5 includes a discussion of the influence of the legacy security on new security solutions. It presents formulas needed for planning evolving security measures. Section 6 includes descriptions of solvable legacy security problems and some solutions.

2. GRADED SECURITY MODEL

Here we briefly introduce variables and functions used in the graded security model. The overall security of a system is described by a *security class*. It shows how the security goals (confidentiality, integrity, availability, ...) are satisfied. It is determined by assigning *security levels* to *security goals*, and is denoted by a respective tuple of pairs, e.g., `C2I1A1M2` for the system that has the second level of confidentiality *C*, the first level of integrity *I* etc.

To achieve the security goals, proper security measures have to be taken. There may be a large number (hundreds) of measures. It is reasonable to group them into security measures groups $g_1, g_2, \ldots, g_n$. The grouping should be done in such a way that measures of one and the same group will always be used for achieving one and the same level of security. One uses a function f that produces a set of required security measures $f(l, g)$ for a given security measures group g and a security level l of the group. A security class determines the required security level for each group of security measures. Let us denote by s a respective function that produces a security level $s(K, g)$ for a group g when the security class is K. An *abstract security profile* is an assignment of security levels $(0, 1, 2, \text{or } 3)$ to each group of security measures. This can be expressed by the tuple $p = (s(K, g_1), s(K, g_2), \ldots, s(K, g_n))$, where p denotes the abstract security profile and the elements of the tuple p are indexed and appear in the tuple in the same order as the groups of security measures $g_1, g_2, \ldots, g_n$ have been indexed. Knowing the cost function $h(l, g)$ that gives the costs r required for implementing security measures of a group g for a level l, one can calculate the costs of implementing a given abstract security profile:

$$costs(p) = \sum_{i=1}^{n} h(l_i, g_i)\,,$$

where $p = (l_1, l_2, \ldots, l_n)$.

The goal is to keep the value $costs(p)$ as low as possible, guaranteeing a required security. It is assumed that by applying security measures, one achieves security goals with some confidence. The security confidence c of a group g that satisfies the security level l is given by a function $e(l, g)$ and it is a numeric value between 0 and 100 for each group of security measures.

3. INTEGRAL SECURITY METRICS

The graded security model uses coarse-grained metrics differentiating three or four security levels for each security goal. To compare security situations in general, one needs a more precise metric that expresses the quality of a security situation by one numeric value. It is reasonable to take into account influences of all security measures on the overall security of the system. The simplest choice would be to calculate the mean security confidence of all groups. However, the influence of groups on the overall security is different. Therefore, the best solution would be to use partial derivatives of the security measure depending on the security confidences of the groups. These derivatives could be used as coefficients of the security confidences when calculating their mean value. Unfortunately, these derivatives are hard to determine. Instead of the derivatives, one can use empirically found weights of the security confidences.

We have introduced a security metric in [4] that evaluates a security situation on the basis of security confidences provided by the security measures groups. We describe the overall security of a system by means of an integrated security metric S that is a *weighted mean security confidence*, called also *integral security confidence*:

$$S = \sum_{i=1}^{n} a_i c_i\,,$$

where c_i is security confidence of the i-th security measures group, a_i is the weight of the i-th group, and

$$\sum_{i=1}^{n} a_i = 1\,.$$

Using a linear combination of security confidences of measures groups is reasonable as long as a security situation does not change too rapidly. (The gradient of the integral security confidence in the space of confidences of security measures groups can be estimated in such a case and its components used as the required coefficients.)

4. VISUALIZING A SECURITY SITUATION

In this section we very briefly present a tool for making calculations on graded security models. This is a software package with a visual language for specifying security situations and problems. The package has been developed on the basis of the visual software development environment CoCoViLa [3], and it has been described in more detail in [4] and [5]. The package includes expert

knowledge for a particular class of security situations. This expert knowledge is usable only for demonstrating the method—it has been taken mainly from [9].

Fig. 1 shows a specification of a security planning problem. The toolbar has buttons for defining components that will constitute a specification. It includes two buttons for defining security measures groups: one for groups with standard values of parameters, and another for groups with parameters defined as inputs. It includes also buttons for defining a security class, for selecting an optimization method and for defining a graphical output. All these components are also visible on the scheme in Fig. 1. This scheme is a specification of a problem for finding a Pareto-optimal solution for a security class `C2I1A1M2` and specific parameters given for two security measures groups: *User training* and *Encryption*. Each security measures group has a pop-up window. This window is shown for the *Encryption* group in Fig. 1.

We use this package for all calculations on the graded security model. The package is extended with new components for solving the legacy security problems described in the following sections, see Sections 5 and 6.

5. LEGACY SECURITY INFLUENCE

The widely used graded security model is based on the assumption that former investments into the security and already existing security situation do not influence the outcome of the investments planned. The former investments are sometimes included in the total amount of investments calculated. These investments may be included with a factor less than one, but this is still a rough approximation. We propose here an approach that more precisely takes into account the already achieved security.

Let us fix a security measures group and consider only one group of security measures here. Then we can use a simplified form of the functions h and e for calculating costs r and security confidence c—without showing explicitly the security measures group:

$$\begin{aligned} r &= h(l)\,, \\ c &= e(l)\,. \end{aligned}$$

We use also a function for calculating security level l for invested costs, which is an inverse function of h:

$$l = h^{-1}(r)\,.$$

We need data for already existing security:

l' – existing level of security,
c' – existing security confidence.

To continue analysis of security investments, we need a function H that calculates the needed additional investments r depending on the existing security level l' and the required security level l:

$$r = H(l, l')\,.$$

It may seem that instead of the function H one can use a function h^* that calculates the required resources for increasing security level by Δl, where $\Delta l = l - l'$:

$$r = h^*(\Delta l)\,.$$

It is easy to see that in the case when no investments in the security have been done before, i.e. when $l' = 0$, the function h^* coincides with the already known function h. However, in the case of $\Delta l = 0$ and $l' > 0$ we have to consider the degradation of security as well—the security level will decrease with time. This shows that the usage of h^* instead of H would be quite a rough approximation.

This analysis is valid for all security measures groups. But in the general model, we have to introduce an argument g (group number) in each function considered here. This gives us the functions:

$$\begin{aligned} r &= H(l, l', g)\,, \\ r &= h^*(\Delta l, g)\,. \end{aligned}$$

These functions should be obtained from expert knowledge.

Another approach would be to use security confidence c instead of security level. These variables are bound by the function e in the graded security model:

$$c = e(l)\,.$$

The relation between costs and security confidence is expressed by the formulas:

$$\begin{aligned} r &= h(e^{-1}(c))\,, \text{ and} \\ c &= e(h^{-1}(r))\,. \end{aligned}$$

Knowing the already achieved security confidence, one can ask to calculate additional investments for achieving the new security confidence (or keeping the required confidence level). This requires the knowledge of a new function E that gives the costs r for achieving required

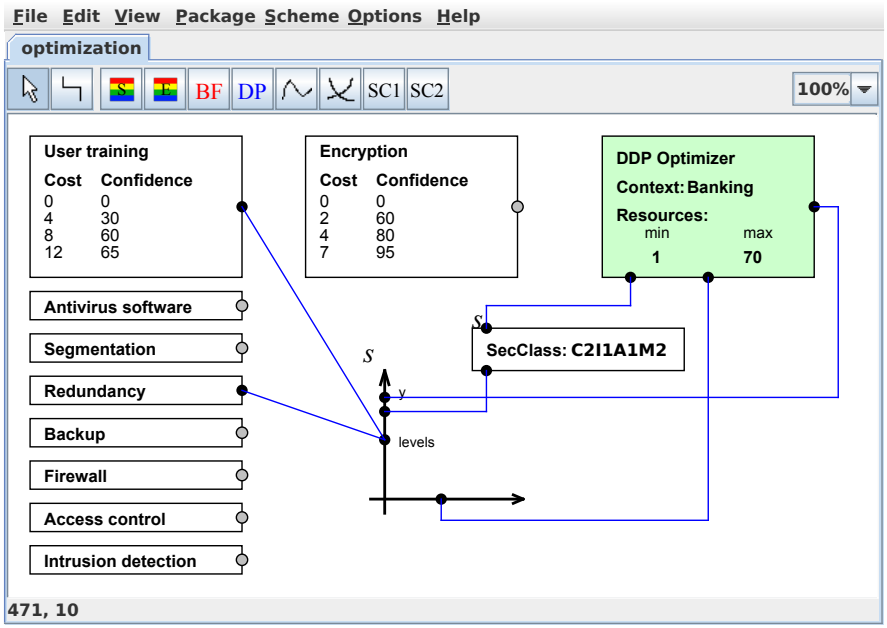


Figure 1. Visual specification of a security situation

security confidence c by upgrading the given security confidence c':

$$r = E(c, c')\,.$$

As discussed above, one can sometimes assume that the costs depend only on the difference Δc of security confidences:

$$\Delta c = c - c'\,,$$

and use the function e^* that calculates the costs:

$$r = e^*(\Delta c)\,.$$

Again, in the general model we have to introduce an argument g (group number) in each function considered here. This gives us the functions for calculating costs in the general case:

$$\begin{aligned} r &= E(c, c', g)\,, \\ r &= e^*(\Delta c, g)\,. \end{aligned}$$

Concluding the analysis here we can say that, for taking into account the legacy security measures in calculating resources required for achieving a given security confidence, we need one of the functions H, h^*, E or e^*. It is preferable to use H or E, because these describe the security situation more precisely. In practice, these functions are represented in a tabular form as expert knowledge. One would like to solve an inverse problem—calculate achievable security confidence for given resources. This is done by using one of the inverse functions H^{-1} or E^{-1} representable by the same tables as H and E:

$$\begin{aligned} l &= H^{-1}(r, l', g)\,, \\ c &= E^{-1}(r, c', g)\,. \end{aligned}$$

Let us call the functions H, h^*, E, e^*, H^{-1} and E^{-1} legacy functions.

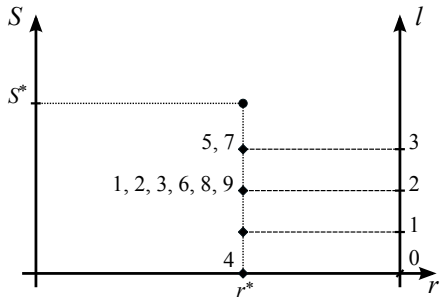
The legacy values of l and r are bound by the functions h and h^{-1} as follows:

$$\begin{aligned} r' &= h(l')\,, \text{ and} \\ l' &= h^{-1}(r')\,. \end{aligned}$$

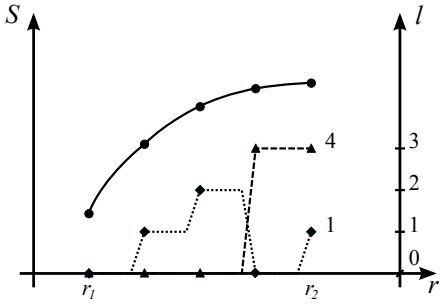
Therefore we can use legacy resources r' instead of l' as inputs of the calculations. We use this in an example in Section 6.

6. OPTIMIZING EVOLVING SECURITY

Security planning can be performed in two different ways. The traditional way is to decide somehow which



(a) Optimal resource assignment for r^*



(b) Pareto-optimal solutions

Figure 2. Solutions of the optimization problem of finding the best assignments of resources to different security measures groups

security levels are required, and to calculate the required resources, using a function H or E. This is an application of the well-known *graded security method* [6]. The security levels are usually prescribed by some standards in this case.

Another way is to solve the *inverse problem*: for given resources find the best assignment of the resources to different security measures groups. This is an optimization problem that can be solved by means of discrete dynamic programming as shown in [5]. The quality of a solution is evaluated by the integral security metric S introduced in [4] and described in Section 3. Fig. 2a shows a solution of the inverse problem: the value of S for given resources r, and also selected security levels of security measures groups. The levels for the groups numbered from 1 to 9 are shown on the right side scale.

Besides the value of S, one may have to consider constraints put on the solution by the security class K, if it is given—all security goals prescribed by K must be satisfied. If priorities are assigned to the security goals, then it is possible to solve a more general problem: find the best possible security solution that satisfies the goal with the highest priority and, if possible, then satisfies also a goal with the next higher priority etc.

Our experiments have shown that the dynamic programming method is fast enough for solving even a more general problem: finding a Pareto-optimal set of security solutions for a given range of resources. Simply speaking, this means that the problem above must be solved for many values of resource r and the result must be plotted as a curve as shown in Fig. 2b.

Fig. 3 shows such a curve for resources from 1 to 70 units. It is obtained by using the expert system described in [4] for the problem specified in Fig. 1. We can see that the security class is `C2I1A1M2` and that two security measures groups (*User training* and *Encryption*) get specific input values for the functions h and e. Other measures groups use the values from the built-in expert system.

In Fig. 3, the lower graphs indicate (on the scale shown on the right) the optimal levels of two measures groups (*Redundancy* and *User training*) corresponding to the given amount of resources. These graphs are not monotonic as can be seen in this example at the resource values 35 and 36. For a more detailed explanation see [5].

Let us consider now the inverse problem considering also the legacy security: given a security class K, resources r, existing security levels l' and a legacy function H^{-1}, find the security solution with the highest value of mean weighted security confidence S that satisfies all security goals of K. This problem may or may not have a solution. Even if it does not have a solution, the problem without the constraint K (without the requirements on security goals) will have a solution. It is interesting to notice that, in the case when the problem has a solution, this solution may be different from the solution obtained without the constraint K.

Fig. 4 shows a solution for both cases: the red curve presents a solution for the problem with a constraint $K =$ `C3I1A1M2`, and the green curve presents a solution for the unrestricted problem. We can see the cases where prescribing K gives worse values of S.

For solving the legacy problems we have extended the expert system by adding the legacy information to the

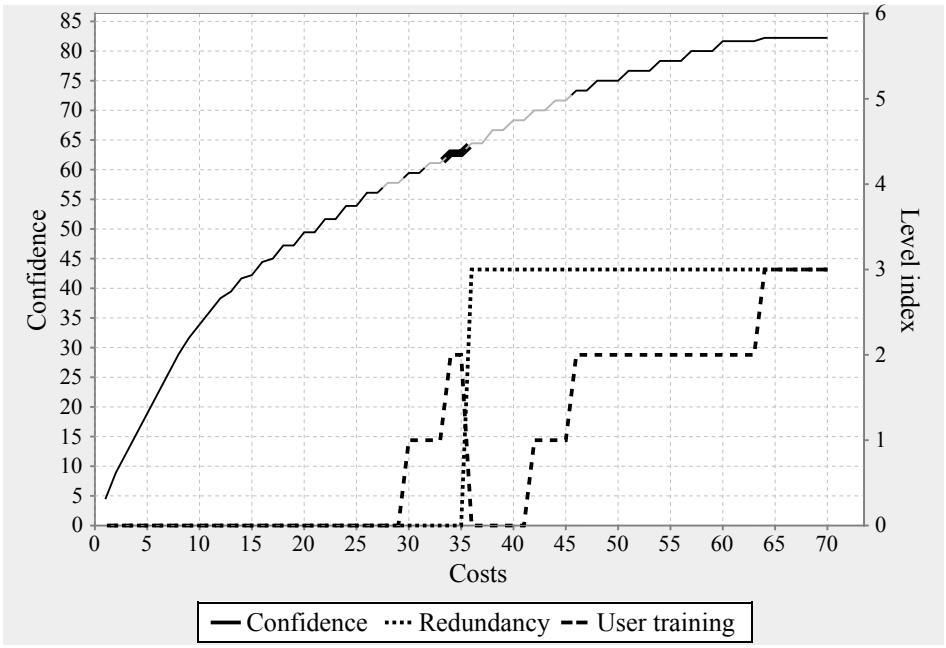


Figure 3. Solution of the problem

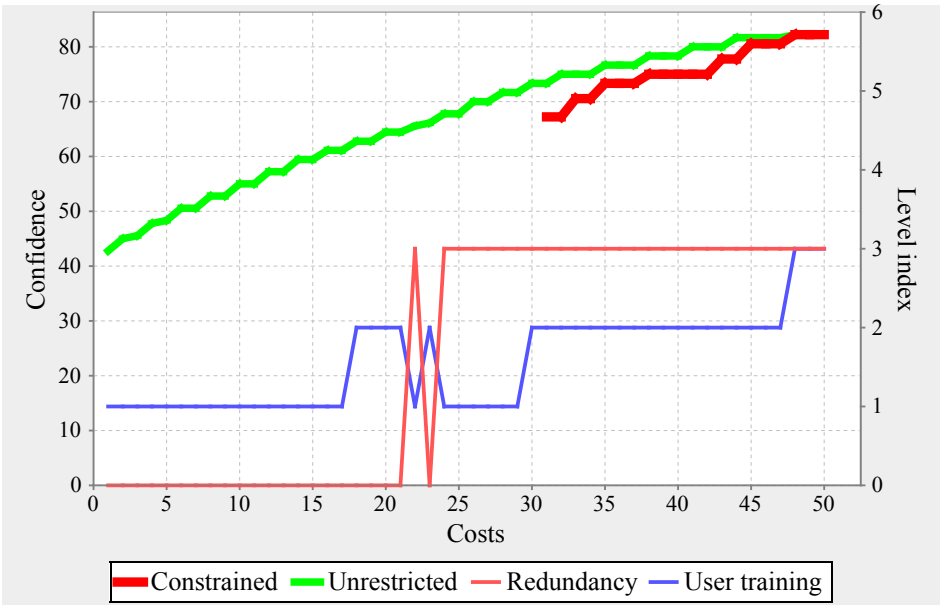


Figure 4. Solutions with and without a constraint

Table 1. Values of legacy resource and decay

g	r'	q
User training	4	0.3
Antivirus software	4	0.6
Segmentation	4	0.2
Redundancy	0	0.3
Backup	4	1.0
Firewall	4	0.5
Access control	4	0.2
Intrusion detection	4	0.5
Encryption	4	0.2

components representing security measures groups, and adding the calculation of the legacy function

$$c = e(h^{-1}(r_0)) \,,$$

where $r_0 = r + (1 - q)r'$ is an effective resource that takes into account both current resource r and decayed value of the legacy resource r'; q is a decay of a resource, $q < 1$. We have used the values of legacy resource r' and decay q given in Table 1.

Knowing the legacy function, we can plan optimal security measures for a number of time intervals (years) in advance. The values l' of existing security levels must be given as initial data. The values of l' for each following year must be taken equal to the values of l of the previous year. The Pareto-optimal set is a surface in a multidimensional space with coordinates $r, y, l_1, \ldots, l_n$ and S, where y is the year number in this case.

Even if we consider Pareto-optimal solutions only for one year, visualization of the Pareto-optimal set is possible only in a special case when all security levels of all security measures groups are equal. In this case, the Pareto-optimal set is a surface in the three-dimensional space r, S, l, where l is the confidence level of all measures groups, and this can be visualized.

7. CONCLUDING REMARKS

The software developed in the present work for analyzing security situations is easy to use for security experts. The developed experimental tool has a simple graphical interface and a visualization component that supports its usage by security managers of all levels. The experiments have also shown that stability of optimal solutions found by the presented method is good. However, the practical applicability of the software will depend on the availability of good expert data representing the legacy function as well as functional dependencies of the graded security model. The developed software has been designed as an expert system. It supports easy inclusion of new expert knowledge, but expert knowledge acquisition is always a complicated task. For specific military communications applications the data must be collected from a log analysis of characteristic attacks and security reports, as well as by the traditional knowledge acquisition means.

Finally, the contemporary security landscape is dynamic and rapidly changing. This is the main reason for developing agile methods of security situation management. The presented method of managing evolving security situations is one of these.

ACKNOWLEDGMENTS

We thank the Cooperative Cyber Defence Centre of Excellence, the Estonian Defence Forces Training and Development Centre of Communication and Information Systems, and the Estonian Ministry of Defence for the support of this work.

REFERENCES

[1] Estonian Informatics Centre. *Estonian Information Systems Three-Level Security Baseline System – ISKE version 4.01.* http://www.ria.ee/27220 (10 Apr 2009).

[2] L. A. Gordon, M. P. Loeb. *Managing Cybersecurity Resources: A Cost-Benefit Analysis.* McGraw-Hill, 2006.

[3] P. Grigorenko, A. Saabas, E. Tyugu. *Visual tool for generative programming*. ACM SIGSOFT Software Engineering Notes, 2005, 30, 5, 249–252.

[4] J. Kivimaa, A. Ojamaa, E. Tyugu. *Graded security expert system.* CRITIS 2008: Third International Workshop on Critical Information Infrastructure Security, Rome, October 13–15 2008. Springer, LNCS, 2009.

[5] A. Ojamaa, E. Tyugu, J. Kivimaa. *Pareto-optimal situation analysis for selection of security measures.* MILCOM 08: Assuring Mission Success: Unclassified Proceedings, San Diego, November 17–19 2008, 7p.

[6] C. E. Pasterczyk. *A graded approach to ISO 9000 implementation for records managers*. Association of Records Managers and Administrators international annual conference, Toronto (Canada), 25–29 September 1994.

[7] *The Common Criteria.* http://www.commoncriteriaportal.org/ (10 Apr 2009)

[8] U. S. Department of Defense. *National Industrial Security Program Operating Manual (NISPOM)*. 2006.

[9] U. S. Department of Defense, Defense Information Systems Agency. *CyberProtect, version 1.1*. July 1999. http://iase.disa.mil/eta/product_description.pdf (10 Apr 2009)

Publication XI

Ojamaa, Andres; Tyugu, Enn (2016).

Enterprise Security Analysis and Training Experience

Critical Information Infrastructures Security: 9th International Conference, CRITIS 2014, Limassol, Cyprus, October 13–15, 2014, Revised Selected Papers. Ed. Panayiotou, C.G.; Ellinas, G.; Kyriakides, E.; Polycarpou, M.M. Cham: Springer, 200–208. (Lecture Notes in Computer Science; 8985).

Enterprise Security Analysis and Training Experience

Andres Ojamaa( ) and Enn Tyugu

Institute of Cybernetics at Tallinn University of Technology,
Akadeemia tee 21, 12618 Tallinn, Estonia
{andres.ojamaa,tyugu}@cs.ioc.ee

Abstract. A holistic approach to security can be introduced by using a model that binds security measures with costs and security metrics. We describe exercises based on the graded security model, and supported by an expert system that are used for training both general managers and security experts. Trainees have to solve a number of problems under conditions that correspond to a realistic critical information infrastructure security planning situation, with the level of details depending on the expertise of trainees.

Keywords: Security training · Graded security · Security model

1 Introduction

We present an approach to teaching managers, where first a small number of basic concepts is explicitly introduced by defining a security model that includes cost considerations as an essential part of the model. These concepts are then used, based on the presented model, in a process that is close to real-life security design situations for making decisions and aided by an expert system. This process includes an abstract analysis section, followed by hands-on training that is usually the favorite usage of the model for the trainees. Performing the analysis of a security model that covers all the essential aspects of cybersecurity in a generalized way helps trainees to get a general picture of the enterprise security.

An expert system for teaching cybersecurity for trainees with different expertise levels was developed, and this expert system was based on a graded security model [4]. It has been used since 2010 in the courses Information and Cyber Security Assurance in Organizations and Foundations and Management of Cyber Security of the international masters program in cybersecurity at Tallinn University of Technology [9].

This paper summarises the basics of rational security design and the graded security model in Sects. 2 and 3, respectively. The security expert system used in the training process is introduced in Sect. 4, followed by the description of the training process in Sect. 5. References to related work are given in Sect. 6.

C.G. Panayiotou et al. (Eds.): CRITIS 2014, LNCS 8985, pp. 200–208, 2016.
DOI: 10.1007/978-3-319-31664-2_21

2 Rational Security Design

Security planning for an enterprise is aimed at finding a distribution of resources for security measures to achieve the best security solution under given conditions. This is called *rational security design*. Inputs for the rational security design are: *total amount of available money (maximal allowed costs), characterization of the secured system, security requirements.*

Finding a security solution assumes using a security metric that offers a possibility of comparing different solutions, and the possibility to express the quality of the solution by a numeric value. We use an *integrated security confidence* for this purpose that takes into account effects of taken security measures (see the following section). Although security planning will be performed as an optimization process, one can use mainly empirical data and expert knowledge. Therefore we call the result a *rational* and not necessarily an optimal solution.

Security planning may require solving several partial problems: evaluating costs required for implementing a selected solution, estimating the relative importance of any taken security solution, detecting similar security measures and collecting them in security measures groups, etc. The present set of exercises is intended to offer hands-on experience for solving these partial problems, as well as for constructing the complete solution.

3 Graded Security Policy and Model

The graded security policy uses a small number of security levels for characterizing each security aspect. As another approximation of reality in the model, we use a small number of security measures groups (about ten), instead of a large number of security measures (hundreds). These simplifications are useful in training, because they make the model comprehensible.

The security model describes how *security measures* are related to *security goals*, what are the *costs* to apply security measures, and what is the *confidence* for guaranteeing the respective *security level*. The concepts used in the graded security model are the following.

3.1 Security Goals

There are three common security goals (or security aspects) that must be achieved: *Confidentiality (C)*, *Integrity (I)*, *Availability (A)*.

Confidentiality guarantees that secured information will be accessible only to actors with the rights for using the information. *Integrity* means that secured information remains intact (can not be corrupted). *Availability* means that information can be accessed/used at any time when needed. It is possible to utilize more security goals—for example, in our expert system, one can use *Mission criticality* as an additional goal.

3.2 Security Class

A security goal can be achieved only to some level, and can practically never reach 100 %. Therefore each security goal has a *security level* that describes how strict the measures that are applied for achieving the goal must be. We use four security levels: *no requirements (0), low security level (1), medium security level (2), high security level (3). Security class* is a tuple of security goals with respective security levels, e.g., *C0I3A3* denotes a security class with confidentiality *(C)* equal to *0*, integrity *(I) 3* and availability *(A) 3*.

3.3 Security Measures Group

A key component of the graded security model is the security measures group (SMG). It unites security measures with similar effects and is described by the parameters: security level l, cost of the security measures c, set of security measures to be taken m and security confidence p. The security level of SMG determines the values of other parameters of the group.

The four parameters l, c, m, p of a group are bound by the four tabulated functions: $c = f_1(l)$, $l = f_2(c)$, $m = f_3(l)$, $p = f_4(l)$ that are different for each group. These functions are given by the expert system, and they depend also on the environment (on a situation) where the security is applied. In our system, SMG as a component of the security model can be used as a black box, because the functions binding its parameters are hidden in the expert system.

A number of security measures groups may vary from one version of a model to another. In the present training environment we have the following groups: *personnel training*, *firewalls*, *encryption*, *antivirus software*, *segmentation*, *redundancy*, *backup*, *access control*, and *intrusion detection*.

The concepts presented here: security goals, security class, security metrics and security measures group constitute the basis of the security education for managers and experts. The security experts, but not the managers, should understand security groups in detail, including functional dependences between their parameters.

3.4 Security Metrics

When security measures are applied, each security measures group i gets a value p_i of *security confidence* that is a percent to which the security is guaranteed by measures of the group (practically always less than 100 %). *Overall security* (integral security confidence) s is a security metric that describes all security aspects of the secured system by means of one single value. This value is calculated as a weighted mean of security confidences of security measures groups:

$$s = \sum_i w_i p_i, \text{ where } \sum_i w_i = 1.$$

The weights w_i characterize importance of each group i in the overall security of system. They depend on a concrete situation and should be assigned by an expert before the usage of the model. Our expert system has also functionality for automatically assigning the weights w_i depending on the type and properties of the secured system.

There are other and more precise ways to define integral security metrics. In particular, the integral security can be calculated analogously to reliability of a system depending on structure of the system [3].

3.5 Security Model

A security model includes a component for each group, and shows also a security class and formulas for calculating costs. Figure 1 in the following section shows a model for 9 security measures groups.

4 Security Expert System

We apply an expert system developed in an earlier work [4] for building and using graded security models based on empirical data from banking practice. Two banks operating in Estonia, Swedbank and SEB, have been interested in this development and have cooperated with us in the development. The expert system is intended for building security models and solving security design problems: calculating the best distribution of given resources, checking reachability of security goals, planning the evolving security for several years, etc. Solving these problems in a training process will be discussed in the next section.

The expert system has been built on an open source visual programming platform CoCoViLa that provides advanced visual interface for applications [8]. Figure 1 shows a window of the expert system for specifying security models. A model is specified by selecting its components from the palette, putting them in the scheme of the model, and adjusting their parameters. Components have default values of parameters, but each of them has a popup window that can be used for adjusting properties of the component. A popup window "Properties" for the security class component is shown on the right side of the figure. It includes a string *C2I1A1M2* that is the security class value.

We can see nine components representing SMGs. Only two of them: *Encryption* and *User training* are presented in the extended way, so that the relation between their costs and confidences is visible. Components for security class and optimizer are also visible in the window. The latter is a control component for calculations, it includes also calculation of total costs and overall security.

There is a component for visualisation of the results, and it is bound with three SMGs: *Antivirus software*, *User training* and *Antivirus software*. Their security levels will be visualised in a results window as it is shown below. Standard connections between the components of the security model (between each group and the security class etc.) are established automatically, and are not visible. The scheme includes also two more components that are needed for adjusting

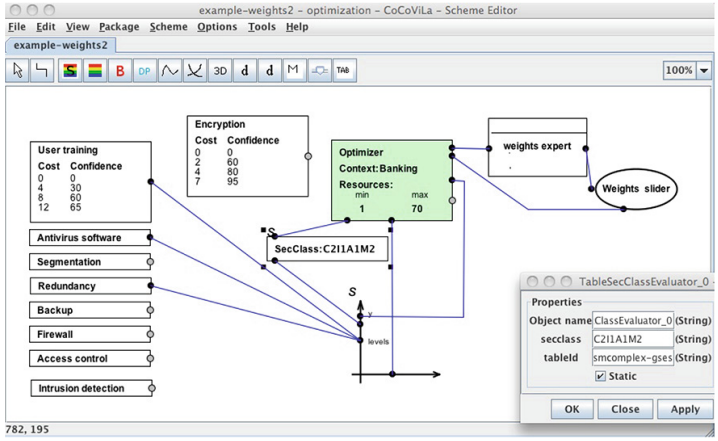


Fig. 1. Visual representation of security model is input for the expert system

the weights w_i for calculating the overall security: *Weights expert* and *Weights slider*.

The scheme in Fig. 1 describes a graded security model of a bank. It is easy to specify different security models, e.g., adding new security measures groups. As we know, the largest model developed so far contains 40 groups [3]. A library of models can be developed for different training situations that can cover essential situations.

The expert system provides full automation of calculations on the specified model. The results created by the visualization component are shown in a window shown in Fig. 2. This window can be viewed as an graphical dashboard representing the whole security situation of an enterprise. It shows the following:

- an overall security curve: the Pareto set of pairs (*costs, overall security*);
- security levels of the SMGs connected to the visualizer component for each value of costs;
- whether the specified security class can be achieved for given costs.

The main result is the topmost curve in the window. It represents a Pareto set consisting of best achievable overall security values for different values of costs. An optimization problem is solved for each point of the curve. This problem is finding a distribution of costs between the SMGs that gives the best value of the overall security for given costs.

The window shown in Fig. 2 includes more curves besides the overall security curve. These curves show the calculated security levels for three SMGs: *Antivirus software*, *Redundancy* and *User training*. To show security levels for some measures group, one has to connect the respective measures group with the *levels* port of the graph object in the scheme as it is shown in Fig. 1.

Colour of the topmost curve gives additional information for each value of costs: red—the requirements of the security class are unsatisfied, yellow—the

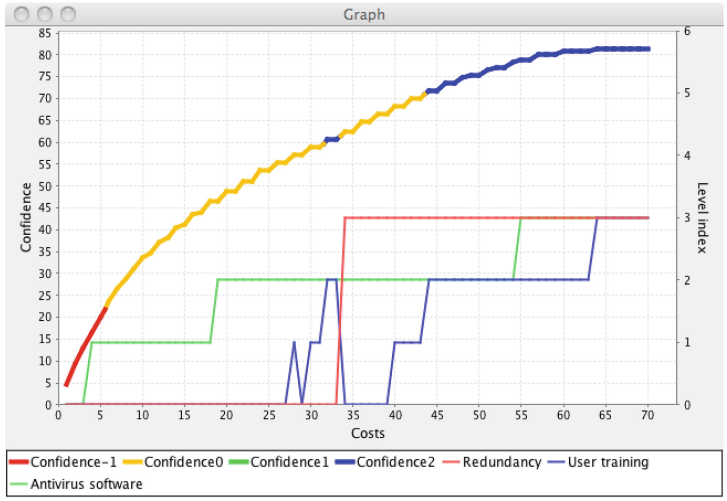


Fig. 2. A results window of the expert system (Color figure online)

requirements of the security class are partially satisfied, and blue—all requirements of the security class are satisfied, i.e., all security goals are achieved at the required levels.

5 Training Process

Trainees are expected to solve a number of security design problems in the conditions that are close to a realistic security planning situation. A suggested order of training steps for these problems is shown in Fig. 3. The first two are introductory steps. The third step—analysis of SMGs is an activity for experts. So are the following two steps of adjustment of parameters. Default parameters given in the expert system can be used in training of general managers. Calculating the best distribution of resources is the main training activity. It is possible to concentrate on this activity immediately after introductory steps. Calculating the evolving security continues the previous step and considers security as a process continuing for several years. Checking the reachability of security goals is an independent problem, but it can be solved by the same means as calculating the best distribution of resources.

If a scheme from the scheme library is used, then the first five training steps can be omitted, and one can immediately start solving the last three problems that are training steps for managers as well as for security experts. Below are described some steps of using the expert system.

The model-based security analysis is suitable for security training of people with different level of experience. It relies on analytic capabilities of a person, and it introduces the basics of security in the form of a security model. The hands-on part of the training is performed using an expert system. The aim of the presented security model and of set of exercises is to help the trainee to

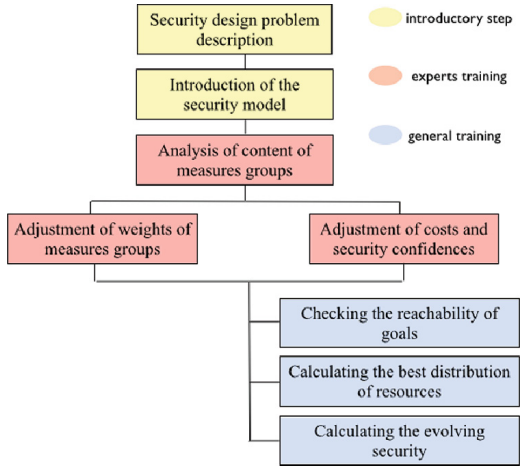


Fig. 3. Order of training steps

build his/her mental model of cybersecurity by combining (1) analytical approach where basic concepts and an explicit security mode are introduced, and (2) hands-on experience by solving security design problems with the help of an expert system.

6 Related Work

Graded security policy is the basis for security protection programs of nuclear security and anti-terrorist security of US Department of Energy [7,12], as well as for several European information assurance standards. In particular, German BSI-Standard 100-2 [2] and Estonian ISKE standard [1] are both based on the graded security policy. We adopt this policy in our expert system, and use the respective security model. Evolving security analysis that we use here has been investigated by Kivimaa et al. [5].

A popular way to teach security concepts is to give some hands-on experience by targeted video games like CyberSiege [10] from the Naval Postgraduate School, or using games from the DISA online training catalogue, e.g., the well-known game CyberProtect [11]. These tools are intended for teaching security concepts for people with varied backgrounds and different levels of expertise. They introduce essential security measures by means of exercises in a more or less realistic situation, but give trainees only implicitly a partial security model. There are also more advanced and complex cyber security learning systems, such as CyberNEXS [6] from Leidos (SAIC). The latter is aimed at development of skills on expert level. CyberNEXS is probably the most advanced of the cybersecurity training systems. It provides games for four scenarios: network defense, forensics, penetration testing (attacks) and capture the flag. We presented here a complementary approach to war games for teaching cybersecurity that is based on the analysis and usage of a security model.

7 Conclusion

We have proposed a model-based technique and software of security training. It relies on analytic capabilities of a person, and it introduces the basics of security in the form of a security model. The hands-on part of the training is performed using an expert system. The aim of the presented security model and of set of exercises is to help the trainee to build his/her mental model of cybersecurity by combining (1) analytical approach where basic concepts and an explicit security mode are introduced, and (2) hands-on experience by solving security design problems with the help of the expert system. Our four years teaching experience on masters level at Tallinn University of Technology [9] has validated the methodology.

Acknowledgements. The authors appreciate the support of the Estonian Academy of Sciences, the target funding project SF0140007s12 of Estonian Ministry of Education and Research, the European Regional Development Fund (ERDF) through Estonian Centre of Excellence in Computer Science (EXCS) project and the project No. 3.2.1201.13-0026 *Model-based Java software development technology.*

References

1. Estonian Information System Authority: Three-level IT baseline security system ISKE. https://www.ria.ee/iske-en
2. German Federal Office for Information Security (BSI): BSI-Standard 100-2 IT Grundschutz Methodology. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/BSIStandards/standard_100-2_e_pdf.pdf
3. Kivimaa, J., Kirt, T.: Evolutionary algorithms for optimal selection of security measures. In: Ottis, R. (ed.) Proceedigs of the 10th European Conferences on Information Warfare and Security, pp. 172–184. Academic Publishers, Reading, UK (2011)
4. Kivimaa, J., Ojamaa, A., Tyugu, E.: Graded security expert system. In: Setola, R., Geretshuber, S. (eds.) CRITIS 2008. LNCS, vol. 5508, pp. 279–286. Springer, Heidelberg (2009)
5. Kivimaa, J., Ojamaa, A., Tyugu, E.: Managing evolving security situations. In: Unclassified Proceedings of the MILCOM 2009, 18–21 October 2009. IEEE, Piscataway (2009)
6. Leidos: CyberNEXS Cyber Security Training. https://www.leidos.com/cybersecurity/solutions/CyberNEXS
7. Lobsenz, G.: DOE Adopts New "Graded" Terrorist Protection Policy (2008). http://pogoarchives.org/m/nss/energydaily-20080826.pdf
8. Modeling and Simulation Group at IoC: CoCoViLa model-based software development platform. http://www.cs.ioc.ee/cocovila/
9. Tallinn University of Technology: Cyber security master's programme. http://www.ttu.ee/studying/masters/masters_programmes/cyber-security/
10. Thompson, M., Irvine, C.: Active learning with the CyberCIEGE video game. In: Proceedings of the 4th Conference on Cyber Security Experimentation and Test, CSET 2011. USENIX Association, Berkeley (2011)

11. U.S. DoD, Defense Information Systems Agency: CyberProtect, version 1.1. http://iase.disa.mil/eta/
12. U.S. DoE, Office of Information Resources: DOE O 470.3B, Graded Security Protection (GSP) Policy. https://www.directives.doe.gov/directives/0470.3-BOrder-b/view

Curriculum Vitae

1. Personal data

Name:	Andres Ojamaa
Date and place of birth:	December 02, 1982, Tallinn, Estonia
Citizenship:	Estonian

2. Contact information

Address:	Akadeemia tee 21, 12618, Tallinn, Estonia
Phone:	+372 620 4223
E-mail:	andres.ojamaa@cs.ioc.ee

3. Education

Educational institution	Graduation year	Education (field of study/degree)
Tallinn University of Technology	2007	Informatics/Master of Science in Engineering
Tallinn University of Technology	2005	Network software and intelligent systems/diploma

4. Language competence/skills (fluent, average, basic skills)

Language	Level
Estonian	native
English	fluent

5. Professional employment

Period	Organisation	Position
2009–...	Institute of Cybernetics at Tallinn University of Technology	researcher
2006–2008	Institute of Cybernetics at Tallinn University of Technology	engineer

6. Scientific work

- Haav, Hele-Mai; Ojamaa, Andres (2016). Semi-automated integration of domain ontologies to DSL meta-models. International Journal of Intelligent Information and Database Systems. [to appear]

- Ojamaa, Andres; Tyugu, Enn (2016). Enterprise security analysis and training experience. Critical Information Infrastructures Security : 9th International Conference, CRITIS 2014, Limassol, Cyprus, October 13-15, 2014, Revised Selected Papers. Ed. Panayiotou, C.G.; Ellinas, G.; Kyriakides, E.; Polycarpou, M.M. Cham: Springer, 200−208. (Lecture Notes in Computer Science; 8985).
- Ojamaa, Andres; Haav, Hele-Mai; Penjam, Jaan (2015). Semi-automated generation of DSL meta models from formal domain ontologies. Model and Data Engineering : 5th International Conference, MEDI 2015, Rhodes, Greece, September 26-28, 2015, Proceedings. Ed. Bellatreche, Ladjel; Manolopoulos, Yannis. Springer, 3−15. (Lecture Notes in Computer Science; 9344).
- Haav, Hele-Mai; Ojamaa, Andres; Grigorenko, Pavel; Kotkas, Vahur (2015). Ontology-based integration of software artefacts for DSL development. On the Move to Meaningful Internet Systems: OTM 2015 Workshops : Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, EI2N, FBM, INBAST, ISDE, META4eS, and MSC 2015, Rhodes, Greece, October 26-30, 2015, Proceedings. Ed. Ciuciu, I. et al. Cham: Springer, 309−318. (Lecture Notes in Computer Science; 9416).
- Ojamaa, Andres; Lind, Uku-Rasmus (2014). Securing Customer Email Communication in E-Commerce. Proceedings 2013 Sixth International Conference on Developments in eSystems Engineering, DeSE 2013 : 16-18 December 2013, Abu Dhabi, UAE. Ed. Hussain, A.; Al Jumeily, D.; Radi, N.; Tawfik, H.; Radvan, R. Los Alamitos, Calif.: IEEE, 291−296.
- Ojamaa, Andres; Kotkas, Vahur; Spichakova, Margarita; Penjam, Jaan (2013). Developing a lean mass customization based manufacturing. 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE 2013), Sydney, Australia, December 3-5, 2013. Piscataway, NJ: IEEE, 28−33.
- Ojamaa, Andres; Düüna, Karl (2012). Assessing the security of Node.js platform. 2012 International Conference for Internet Technology and Secured Transactions (ICITST-2012) : 10-12 Dec.2012, [London, UK]. Piscataway, NJ: IEEE, 348−355.
- Ojamaa, Andres; Düüna, Karl (2012). Security Assessment of Node.js Platform. Information Systems Security : 8th International Conference, ICISS 2012, Guwahati, India, December 15-19, 2012, Proceedings. Ed. Venkatakrishnan, Venkat; Goswami, Diganta. 35−43. (Lecture Notes in Computer Science; 7671).
- Kotkas, Vahur; Ojamaa, Andres; Grigorenko, Pavel; Maigre, Riina; Harf, Mait; Tyugu, Enn (2011). CoCoViLa as a multifunctional simulation platform. Proceedings of the 4th International ICST Conference on Simulation

Tools and Techniques : 21-25 March 2011, Barcelona, Spain, SIMUTools 2011. Brussels: ICST, 195−205.

- Klein, Gabriel; Ojamaa, Andres; Grigorenko, Pavel; Jahnke, Marko; Tyugu, Enn (2010). Enhancing response selection in impact estimation approaches. Concepts and Implementations for Innovative Military Communications and Information Technologies. Ed. Amanowicz, Marek. Warsaw: Military University of Technology, 277−286.
- Kivimaa, Jyri; Ojamaa, Andres; Tyugu, Enn (2009). Managing evolving security situations. MILCOM 2009 : Unclassified Proceedings, October 18-21, 2009, Boston, MA. Piscataway, NJ: IEEE, 1−7.
- Kivimaa, Jüri; Ojamaa, Andres; Tyugu, Enn (2009). Graded security expert system. Critical Information Infrastructures Security : Third International Workshop, CRITIS 2008, Rome, Italy, October 13-15, 2008, Revised Papers. Ed. Setola, Roberto; Geretshuber, Stefan. Berlin: Springer, 279−286. (Lecture Notes in Computer Science; 5508).
- Ojamaa, Andres (2009). Hybrid simulation of large networks. Proceedings of the 2009 International Conference on Modeling, Simulation & Visualization Methods, MSV 2009. Ed. Arabnia, Hamid R.; Deligiannidis, Leonidas. Las Vegas: CSREA Press, 219−225.
- Ojamaa, Andres; Tyugu, Enn; Kivimaa, Jyri (2008). Pareto-optimal situation analysis for selection of security measures. MILCOM 08 : Assuring Mission Success : Unclassified Proceedings, November 17-19 San Diego. 3224−3230.
- Kaur, Kaiko; Ojamaa, Andres (2008). Service oriented database interface for exchanging multi-format tabular data. Databases and Information Systems : Proceedings of the Eighth International Baltic Conference, Baltic DB&IS 2008, Tallinn, June 2-5, 2008. Ed. Haav, Hele-Mai; Kalja, Ahto. Tallinn: Tallinn University of Technology Press, 289−300.
- Kivimaa, Jüri; Ojamaa, Andres; Tyugu, Enn (2008). Graded security expert system. CRITIS 2008 : Third International Workshop on Critical Information Infrastructure Security, Villa Mondragone, Monte Porzio Catone, Rome, October, 13-15, 2008, (Pre-Proceedings). AIIC, ENEA, 333−339.
- Ojamaa, Andres; Tyugu, Enn (2007). Rich components of extendable simulation platform. Proceedings of the 2007 International Conference on Modeling, Simulation & Visualization Methods, MSV 2007 : June 25-28 2007, Las Vegas Nevada, USA. Ed. Arabnia,Hamid R. Las Vegas: CSREA Press, 121−127.
- Eppendahl, A.; Ojamaa, A. (2006). Seeing empty space in an unknown environment without silhouettes. Proceedings of the 3rd International

Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005). Ed. Murase,K.; Sekiyama,K.; Kubota,N.; Naniwa,T.; Sitte,J. Berlin: Springer, 27−32.

7. Defended theses

   - Modular Simulation Platform. Institute of Cybernetics at Tallinn University of Technology, 2007. Supervisor Enn Tõugu.
   - Seeing empty space in real time with a mobile camera using a light field model. Tallinn University of Technology, Department of Computer Science, 2005. Supervisors Adam Eppendahl, Juhan Ernits.

8. Main areas of scientific work/Current research topics

   Cyber security, software engineering, algorithms, computer simulations, computer vision.

Elulookirjeldus

1. Isikuandmed

Ees- ja perekonnanimi:	Andres Ojamaa
Sünniaeg ja -koht:	2. detsember 1982, Tallinn, Eesti
Kodakondsus:	Eesti

2. Kontaktandmed

Aadress:	Akadeemia tee 21, 12618 Tallinn
Telefon:	620 4223
E-posti aadress:	andres.ojamaa@cs.ioc.ee

3. Hariduskäik

Õppeasutus	Lõpetamise aeg	Haridus (eriala/kraad)
Tallinna Tehnikaülikool	2007	informaatika/tehnikateaduste magister
Tallinna Tehnikaülikool	2005	võrgutarkvara ja intelligentsed süsteemid/diplom

4. Keelteoskus (alg-, kesk- või kõrgtase)

Keel	Tase
eesti keel	emakeel
inglise keel	kõrgtase

5. Teenistuskäik

Töötamise aeg	Tööandja nimetus	Ametikoht
2009–...	Tallinna Tehnikaülikooli Küberneetika instituut	teadur
2006–2008	Tallinna Tehnikaülikooli Küberneetika instituut	insener

6. Teadustegevus

- Haav, Hele-Mai; Ojamaa, Andres (2016). Semi-automated integration of domain ontologies to DSL meta-models. International Journal of Intelligent Information and Database Systems. [to appear]

- Ojamaa, Andres; Tyugu, Enn (2016). Enterprise security analysis and training experience. Critical Information Infrastructures Security : 9th International Conference, CRITIS 2014, Limassol, Cyprus, October 13-15, 2014, Revised Selected Papers. Ed. Panayiotou, C.G.; Ellinas, G.; Kyriakides, E.; Polycarpou, M.M. Cham: Springer, 200−208. (Lecture Notes in Computer Science; 8985).
- Ojamaa, Andres; Haav, Hele-Mai; Penjam, Jaan (2015). Semi-automated generation of DSL meta models from formal domain ontologies. Model and Data Engineering : 5th International Conference, MEDI 2015, Rhodes, Greece, September 26-28, 2015, Proceedings. Ed. Bellatreche, Ladjel; Manolopoulos, Yannis. Springer, 3−15. (Lecture Notes in Computer Science; 9344).
- Haav, Hele-Mai; Ojamaa, Andres; Grigorenko, Pavel; Kotkas, Vahur (2015). Ontology-based integration of software artefacts for DSL development. On the Move to Meaningful Internet Systems: OTM 2015 Workshops : Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, EI2N, FBM, INBAST, ISDE, META4eS, and MSC 2015, Rhodes, Greece, October 26-30, 2015, Proceedings. Ed. Ciuciu, I. et al. Cham: Springer, 309−318. (Lecture Notes in Computer Science; 9416).
- Ojamaa, Andres; Lind, Uku-Rasmus (2014). Securing Customer Email Communication in E-Commerce. Proceedings 2013 Sixth International Conference on Developments in eSystems Engineering, DeSE 2013 : 16-18 December 2013, Abu Dhabi, UAE. Ed. Hussain, A.; Al Jumeily, D.; Radi, N.; Tawfik, H.; Radvan, R. Los Alamitos, Calif.: IEEE, 291−296.
- Ojamaa, Andres; Kotkas, Vahur; Spichakova, Margarita; Penjam, Jaan (2013). Developing a lean mass customization based manufacturing. 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE 2013), Sydney, Australia, December 3-5, 2013. Piscataway, NJ: IEEE, 28−33.
- Ojamaa, Andres; Düüna, Karl (2012). Assessing the security of Node.js platform. 2012 International Conference for Internet Technology and Secured Transactions (ICITST-2012) : 10-12 Dec.2012, [London, UK]. Piscataway, NJ: IEEE, 348−355.
- Ojamaa, Andres; Düüna, Karl (2012). Security Assessment of Node.js Platform. Information Systems Security : 8th International Conference, ICISS 2012, Guwahati, India, December 15-19, 2012, Proceedings. Ed. Venkatakrishnan, Venkat; Goswami, Diganta. 35−43. (Lecture Notes in Computer Science; 7671).
- Kotkas, Vahur; Ojamaa, Andres; Grigorenko, Pavel; Maigre, Riina; Harf, Mait; Tyugu, Enn (2011). CoCoViLa as a multifunctional simulation platform. Proceedings of the 4th International ICST Conference on Simulation

Tools and Techniques : 21-25 March 2011, Barcelona, Spain, SIMUTools 2011. Brussels: ICST, 195−205.

- Klein, Gabriel; Ojamaa, Andres; Grigorenko, Pavel; Jahnke, Marko; Tyugu, Enn (2010). Enhancing response selection in impact estimation approaches. Concepts and Implementations for Innovative Military Communications and Information Technologies. Ed. Amanowicz, Marek. Warsaw: Military University of Technology, 277−286.
- Kivimaa, Jyri; Ojamaa, Andres; Tyugu, Enn (2009). Managing evolving security situations. MILCOM 2009 : Unclassified Proceedings, October 18-21, 2009, Boston, MA. Piscataway, NJ: IEEE, 1−7.
- Kivimaa, Jüri; Ojamaa, Andres; Tyugu, Enn (2009). Graded security expert system. Critical Information Infrastructures Security : Third International Workshop, CRITIS 2008, Rome, Italy, October 13-15, 2008, Revised Papers. Ed. Setola, Roberto; Geretshuber, Stefan. Berlin: Springer, 279−286. (Lecture Notes in Computer Science; 5508).
- Ojamaa, Andres (2009). Hybrid simulation of large networks. Proceedings of the 2009 International Conference on Modeling, Simulation & Visualization Methods, MSV 2009. Ed. Arabnia, Hamid R.; Deligiannidis, Leonidas. Las Vegas: CSREA Press, 219−225.
- Ojamaa, Andres; Tyugu, Enn; Kivimaa, Jyri (2008). Pareto-optimal situation analysis for selection of security measures. MILCOM 08 : Assuring Mission Success : Unclassified Proceedings, November 17-19 San Diego. 3224−3230.
- Kaur, Kaiko; Ojamaa, Andres (2008). Service oriented database interface for exchanging multi-format tabular data. Databases and Information Systems : Proceedings of the Eighth International Baltic Conference, Baltic DB&IS 2008, Tallinn, June 2-5, 2008. Ed. Haav, Hele-Mai; Kalja, Ahto. Tallinn: Tallinn University of Technology Press, 289−300.
- Kivimaa, Jüri; Ojamaa, Andres; Tyugu, Enn (2008). Graded security expert system. CRITIS 2008 : Third International Workshop on Critical Information Infrastructure Security, Villa Mondragone, Monte Porzio Catone, Rome, October, 13-15, 2008, (Pre-Proceedings). AIIC, ENEA, 333−339.
- Ojamaa, Andres; Tyugu, Enn (2007). Rich components of extendable simulation platform. Proceedings of the 2007 International Conference on Modeling, Simulation & Visualization Methods, MSV 2007 : June 25-28 2007, Las Vegas Nevada, USA. Ed. Arabnia,Hamid R. Las Vegas: CSREA Press, 121−127.
- Eppendahl, A.; Ojamaa, A. (2006). Seeing empty space in an unknown environment without silhouettes. Proceedings of the 3rd International

Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005). Ed. Murase,K.; Sekiyama,K.; Kubota,N.; Naniwa,T.; Sitte,J. Berlin: Springer, 27−32.

7. Kaitstud lõputööd

   - Modulaarne simuleerimisplatvorm, Tallinna Tehnikaülikool, Küberneetika Instituut, 2007, juhendaja Enn Tõugu.
   - Valgusväljamudeli abil reaalajas tühja ruumi nägemine kasutades liikuvat kaamerat. Tallinna Tehnikaülikool, Arvutiteaduse Instituut, 2005, juhendajad Adam Eppendahl, Juhan Ernits.

8. Teadustöö põhisuunad

   Küberturve, tarkvaratehnika, algoritmid, arvutisimulatsioonid, masinnägemine.

DISSERTATIONS DEFENDED AT TALLINN UNIVERSITY OF TECHNOLOGY ON *INFORMATICS AND SYSTEM ENGINEERING*

1. **Lea Elmik**. Informational Modelling of a Communication Office. 1992.

2. **Kalle Tammemäe**. Control Intensive Digital System Synthesis. 1997.

3. **Eerik Lossmann**. Complex Signal Classification Algorithms, Based on the Third-Order Statistical Models. 1999.

4. **Kaido Kikkas**. Using the Internet in Rehabilitation of People with Mobility Impairments – Case Studies and Views from Estonia. 1999.

5. **Nazmun Nahar**. Global Electronic Commerce Process: Business-to-Business. 1999.

6. **Jevgeni Riipulk**. Microwave Radiometry for Medical Applications. 2000.

7. **Alar Kuusik**. Compact Smart Home Systems: Design and Verification of Cost Effective Hardware Solutions. 2001.

8. **Jaan Raik**. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams. 2001.

9. **Andri Riid**. Transparent Fuzzy Systems: Model and Control. 2002.

10. **Marina Brik**. Investigation and Development of Test Generation Methods for Control Part of Digital Systems. 2002.

11. **Raul Land**. Synchronous Approximation and Processing of Sampled Data Signals. 2002.

12. **Ants Ronk**. An Extended Block-Adaptive Fourier Analyser for Analysis and Reproduction of Periodic Components of Band-Limited Discrete-Time Signals. 2002.

13. **Toivo Paavle**. System Level Modeling of the Phase Locked Loops: Behavioral Analysis and Parameterization. 2003.

14. **Irina Astrova**. On Integration of Object-Oriented Applications with Relational Databases. 2003.

15. **Kuldar Taveter**. A Multi-Perspective Methodology for Agent-Oriented Business Modelling and Simulation. 2004.

16. **Taivo Kangilaski**. Eesti Energia käiduhaldussüsteem. 2004.

17. **Artur Jutman**. Selected Issues of Modeling, Verification and Testing of Digital Systems. 2004.

18. **Ander Tenno**. Simulation and Estimation of Electro-Chemical Processes in Maintenance-Free Batteries with Fixed Electrolyte. 2004.

19. **Oleg Korolkov**. Formation of Diffusion Welded Al Contacts to Semiconductor Silicon. 2004.

20. **Risto Vaarandi**. Tools and Techniques for Event Log Analysis. 2005.

21. **Marko Koort**. Transmitter Power Control in Wireless Communication Systems. 2005.

22. **Raul Savimaa**. Modelling Emergent Behaviour of Organizations. Time-Aware, UML and Agent Based Approach. 2005.

23. **Raido Kurel**. Investigation of Electrical Characteristics of SiC Based Complementary JBS Structures. 2005.

24. **Rainer Taniloo**. Ökonoomsete negatiivse diferentsiaaltakistusega astmete ja elementide disainimine ja optimeerimine. 2005.

25. **Pauli Lallo**. Adaptive Secure Data Transmission Method for OSI Level I. 2005.

26. **Deniss Kumlander**. Some Practical Algorithms to Solve the Maximum Clique Problem. 2005.

27. **Tarmo Veskioja**. Stable Marriage Problem and College Admission. 2005.

28. **Elena Fomina**. Low Power Finite State Machine Synthesis. 2005.

29. **Eero Ivask**. Digital Test in WEB-Based Environment 2006.

30. **Виктор Войтович**. Разработка технологий выращивания из жидкой фазы эпитаксиальных структур арсенида галлия с высоковольтным p-n переходом и изготовления диодов на их основе. 2006.

31. **Tanel Alumäe**. Methods for Estonian Large Vocabulary Speech Recognition. 2006.

32. **Erki Eessaar**. Relational and Object-Relational Database Management Systems as Platforms for Managing Softwareengineering Artefacts. 2006.

33. **Rauno Gordon**. Modelling of Cardiac Dynamics and Intracardiac Bio-impedance. 2007.

34. **Madis Listak**. A Task-Oriented Design of a Biologically Inspired Underwater Robot. 2007.

35. **Elmet Orasson**. Hybrid Built-in Self-Test. Methods and Tools for Analysis and Optimization of BIST. 2007.

36. **Eduard Petlenkov**. Neural Networks Based Identification and Control of Nonlinear Systems: ANARX Model Based Approach. 2007.

37. **Toomas Kirt**. Concept Formation in Exploratory Data Analysis: Case Studies of Linguistic and Banking Data. 2007.

38. **Juhan-Peep Ernits**. Two State Space Reduction Techniques for Explicit State Model Checking. 2007.

39. **Innar Liiv**. Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management. 2008.

40. **Andrei Pokatilov**. Development of National Standard for Voltage Unit Based on Solid-State References. 2008.

41. **Karin Lindroos**. Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments. 2008.

42. **Maksim Jenihhin**. Simulation-Based Hardware Verification with High-Level Decision Diagrams. 2008.

43. **Ando Saabas**. Logics for Low-Level Code and Proof-Preserving Program Transformations. 2008.

44. **Ilja Tšahhirov**. Security Protocols Analysis in the Computational Model – Dependency Flow Graphs-Based Approach. 2008.

45. **Toomas Ruuben**. Wideband Digital Beamforming in Sonar Systems. 2009.

46. **Sergei Devadze**. Fault Simulation of Digital Systems. 2009.

47. **Andrei Krivošei**. Model Based Method for Adaptive Decomposition of the Thoracic Bio-Impedance Variations into Cardiac and Respiratory Components. 2009.

48. **Vineeth Govind**. DfT-Based External Test and Diagnosis of Mesh-like Networks on Chips. 2009.

49. **Andres Kull**. Model-Based Testing of Reactive Systems. 2009.

50. **Ants Torim**. Formal Concepts in the Theory of Monotone Systems. 2009.

51. **Erika Matsak**. Discovering Logical Constructs from Estonian Children Language. 2009.

52. **Paul Annus**. Multichannel Bioimpedance Spectroscopy: Instrumentation Methods and Design Principles. 2009.

53. **Maris Tõnso**. Computer Algebra Tools for Modelling, Analysis and Synthesis for Nonlinear Control Systems. 2010.

54. **Aivo Jürgenson**. Efficient Semantics of Parallel and Serial Models of Attack Trees. 2010.

55. **Erkki Joasoon**. The Tactile Feedback Device for Multi-Touch User Interfaces. 2010.

56. **Jürgo-Sören Preden**. Enhancing Situation – Awareness Cognition and Reasoning of Ad-Hoc Network Agents. 2010.

57. **Pavel Grigorenko**. Higher-Order Attribute Semantics of Flat Languages. 2010.

58. **Anna Rannaste**. Hierarcical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits. 2010.

59. **Sergei Strik**. Battery Charging and Full-Featured Battery Charger Integrated Circuit for Portable Applications. 2011.

60. **Rain Ottis**. A Systematic Approach to Offensive Volunteer Cyber Militia. 2011.

61. **Natalja Sleptšuk**. Investigation of the Intermediate Layer in the Metal-Silicon Carbide Contact Obtained by Diffusion Welding. 2011.

62. **Martin Jaanus**. The Interactive Learning Environment for Mobile Laboratories. 2011.

63. **Argo Kasemaa**. Analog Front End Components for Bio-Impedance Measurement: Current Source Design and Implementation. 2011.

64. **Kenneth Geers**. Strategic Cyber Security: Evaluating Nation-State Cyber Attack Mitigation Strategies. 2011.

65. **Riina Maigre**. Composition of Web Services on Large Service Models. 2011.

66. **Helena Kruus**. Optimization of Built-in Self-Test in Digital Systems. 2011.

67. **Gunnar Piho**. Archetypes Based Techniques for Development of Domains, Requirements and Sofware. 2011.

68. **Juri Gavšin**. Intrinsic Robot Safety Through Reversibility of Actions. 2011.

69. **Dmitri Mihhailov**. Hardware Implementation of Recursive Sorting Algorithms Using Tree-like Structures and HFSM Models. 2012.

70. **Anton Tšertov**. System Modeling for Processor-Centric Test Automation. 2012.

71. **Sergei Kostin**. Self-Diagnosis in Digital Systems. 2012.

72. **Mihkel Tagel**. System-Level Design of Timing-Sensitive Network-on-Chip Based Dependable Systems. 2012.

73. **Juri Belikov**. Polynomial Methods for Nonlinear Control Systems. 2012.

74. **Kristina Vassiljeva**. Restricted Connectivity Neural Networks based Identification for Control. 2012.

75. **Tarmo Robal**. Towards Adaptive Web – Analysing and Recommending Web Users` Behaviour. 2012.

76. **Anton Karputkin**. Formal Verification and Error Correction on High-Level Decision Diagrams. 2012.

77. **Vadim Kimlaychuk**. Simulations in Multi-Agent Communication System. 2012.

78. **Taavi Viilukas**. Constraints Solving Based Hierarchical Test Generation for Synchronous Sequential Circuits. 2012.

79. **Marko Kääramees**. A Symbolic Approach to Model-based Online Testing. 2012.

80. **Enar Reilent**. Whiteboard Architecture for the Multi-agent Sensor Systems. 2012.

81. **Jaan Ojarand**. Wideband Excitation Signals for Fast Impedance Spectroscopy of Biological Objects. 2012.

82. **Igor Aleksejev**. FPGA-based Embedded Virtual Instrumentation. 2013.

83. **Juri Mihhailov**. Accurate Flexible Current Measurement Method and its Realization in Power and Battery Management Integrated Circuits for Portable Applications. 2013.

84. **Tõnis Saar**. The Piezo-Electric Impedance Spectroscopy: Solutions and Applications. 2013.

85. **Ermo Täks**. An Automated Legal Content Capture and Visualisation Method. 2013.

86. **Uljana Reinsalu**. Fault Simulation and Code Coverage Analysis of RTL Designs Using High-Level Decision Diagrams. 2013.

87. **Anton Tšepurov**. Hardware Modeling for Design Verification and Debug. 2013.

88. **Ivo Müürsepp**. Robust Detectors for Cognitive Radio. 2013.

89. **Jaas Ježov**. Pressure sensitive lateral line for underwater robot. 2013.

90. **Vadim Kaparin**. Transformation of Nonlinear State Equations into Observer Form. 2013.

92. **Reeno Reeder**. Development and Optimisation of Modelling Methods and Algorithms for Terahertz Range Radiation Sources Based on Quantum Well Heterostructures. 2014.

93. **Ants Koel**. GaAs and SiC Semiconductor Materials Based Power Structures: Static and Dynamic Behavior Analysis. 2014.

94. **Jaan Übi**. Methods for Coopetition and Retention Analysis: An Application to University Management. 2014.

95. **Innokenti Sobolev**. Hyperspectral Data Processing and Interpretation in Remote Sensing Based on Laser-Induced Fluorescence Method. 2014.

96. **Jana Toompuu**. Investigation of the Specific Deep Levels in *p*-, *i*- and *n*-Regions of GaAs p^+-*pin*-n^+ Structures. 2014.

97. **Taavi Salumäe**. Flow-Sensitive Robotic Fish: From Concept to Experiments. 2015.

98. **Yar Muhammad**. A Parametric Framework for Modelling of Bioelectrical Signals. 2015.

99. **Ago Mõlder**. Image Processing Solutions for Precise Road Profile Measurement Systems. 2015.

100. **Kairit Sirts**. Non-Parametric Bayesian Models for Computational Morphology. 2015.

101. **Alina Gavrijaševa**. Coin Validation by Electromagnetic, Acoustic and Visual Features. 2015.

102. **Emiliano Pastorelli**. Analysis and 3D Visualisation of Microstructured Materials on Custom-Built Virtual Reality Environment. 2015.

103. **Asko Ristolainen**. Phantom Organs and their Applications in Robotic Surgery and Radiology Training. 2015.

104. **Aleksei Tepljakov**. Fractional-order Modeling and Control of Dynamic Systems. 2015.

105. **Ahti Lohk**. A System of Test Patterns to Check and Validate the Semantic Hierarchies of Wordnet-type Dictionaries. 2015.

106. **Hanno Hantson**. Mutation-Based Verification and Error Correction in High-Level Designs. 2015.

107. **Lin Li**. Statistical Methods for Ultrasound Image Segmentation. 2015.

108. **Aleksandr Lenin**. Reliable and Efficient Determination of the Likelihood of Rational Attacks. 2015.

109. **Maksim Gorev**. At-Speed Testing and Test Quality Evaluation for High-Performance Pipelined Systems. 2016.

110. **Mari-Anne Meister**. Electromagnetic Environment and Propagation Factors of Short-Wave Range in Estonia. 2016.

111. **Syed Saif Abrar**. Comprehensive Abstraction of VHDL RTL Cores to ESL SystemC. 2016.

112. **Arvo Kaldmäe**. Advanced Design of Nonlinear Discrete-time and Delayed Systems. 2016.

113. **Mairo Leier**. Scalable Open Platform for Reliable Medical Sensorics. 2016.

114. **Georgios Giannoukos**. Mathematical and Physical Modelling of Dynamic Electrical Impedance. 2016.

115. **Aivo Anier**. Model Based Framework for Distributed Control and Testing of Cyber-Physical Systems. 2016.

116. **Denis Firsov**. Certification of Context-Free Grammar Algorithms. 2016.

117. **Sergei Astatpov**. Distributed Signal Processing for Situation Assessment in Cyber-Physical Systems. 2016.

118. **Erkki Moorits**. Embedded Software Solutions for Development of Marine Navigation Light Systems. 2016.