TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Timo Vepsäläinen

Model-Driven Development of Control Applications: On Modeling Tools, Simulations and Safety

Julkaisu 1303 • Publication 1303

Tampere 2015

Tampereen teknillinen yliopisto. Julkaisu 1303
Tampere University of Technology. Publication 1303

Timo Vepsäläinen

Model-Driven Development of Control Applications:
On Modeling Tools, Simulations and Safety

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Festia Building, Auditorium Pieni Sali 1, at Tampere University of Technology, on the 5th of June 2015, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2015

ISBN 978-952-15-3528-4 (printed)
ISBN 978-952-15-3536-9 (PDF)
ISSN 1459-2045

Abstract

Control systems are required in various industrial applications varying from individual machines to manufacturing plants and enterprises. Software applications have an important role as an implementation technology in such systems, which can be based on Distributed Control System (DCS) or Programmable Control System (PLC) platforms, for example. Control applications are computer programs that, with control system hardware, perform control tasks. Control applications are efficient and flexible by nature; however, their development is a complex task that requires the collaboration of experts and information from various domains of expertise.

This thesis studies the use of Model-Driven Development (MDD) techniques in control application development. MDD is a software development methodology in which models are used as primary engineering artefacts and processed with both manual work and automated model transformations. The objective of the thesis is to explore whether or not control application development can benefit from MDD and selected technologies enabled by it. The research methodology followed in the thesis is the constructive approach of design science.

To answer the research questions, tools are developed for modeling and developing control applications using UML Automation Profile (UML AP) in a model-driven development process. The modeling approach is developed based on open source tools on Eclipse platform. In the approach, modeling concepts are kept extendable. Models can be processed with model transformation techniques that plug in to the tool. The approach takes into account domain requirements related to, for example, re-use of design. According to assessment of industrial applicability of the approach and tools as part of it, they could be used for developing industrial DCS based control applications.

Simulation approaches that can be used in conjunction to model-driven development of control applications are presented and compared. Development of a model-in-the-loop simulation support is rationalized to enable the use of simulations early while taking into account the special characteristics of the domain. A simulator integration is developed that transforms UML AP control application models to Modelica Modeling Language (ModelicaML) models, thus enabling closed-loop simulations with ModelicaML models of plants to be controlled. The simulation approach is applied successfully in simulations of machinery applications and process industry processes.

Model-driven development of safety applications, which are parts of safety systems, would require taking into account safety standard requirements related to modeling

techniques and documentation, for example. Related to this aspect, the thesis focuses on extending the information content of models with aspects that are required for safety applications. The modeling of hazards and their associated risks is supported with fault tree notation. The risk and hazard information is integrated into the development process in order to improve traceability. Automated functions enable generating documentation and performing consistency checks related to the use of standard solutions, for example. When applicable, techniques and notations, such as logic diagrams, have been chosen so that they are intuitive to developers but also comply with recommendations of safety standards.

Keywords: control application, model-driven development, modeling, simulation, safety

Preface

The research, on which this thesis builds upon, has been carried out at the Department of Automation Science and Engineering at the Tampere University of Technology (TUT). The research work was conducted between 2008 and 2014 as part of several joint-funded research projects in collaboration with Tekes the Finnish Funding Agency for Innovation, other research institutes as well as both Finnish and international companies. During the years 2012-2015, the funding of TUT President's Doctoral Program made it possible to focus on the thesis work with less project pressure.

For being able to finish this thesis, I must thank several people. First and foremost, I would like to thank the supervisor of my studies, Professor Seppo Kuikka, for the possibility to work in the Automation Software research group and for all the guidance during the work. Without the support and trust, completing this thesis would not have been possible.

I want to thank Professor Georg Frey from Saarland University and Research Professor Tommi Karhela from the VTT Technical Research Centre of Finland for the preliminary examination of the thesis and for the remarks and feedback. I want to thank Professor Georg Frey and Professor Kauko Leiviskä from the University of Oulu for acting as opponents in the public examination.

I would like to thank the professors, colleagues and the personnel of the Department of Automation Science and Engineering for the pleasant and motivating working environment. Especially I would like to thank the members of the Automation Software research group with whom I have had the pleasure to work. I want to thank Dr. David Hästbacka, Lic.Sc. Outi Rask, Jari Rauhamäki and Petri Kannisto but also the former members of the group. I would also like to thank the colleagues and friends from Aalto University and VTT Technical Research Centre of Finland. With many of these people, I have also had the honor to co-author publications.

Thanks also to my friends for helping me forget the work and studies every once in a while. Finally, I am grateful to my parents for their support and encouragement during my M.Sc. and doctoral studies that turned out to take quite some time.

Lempäälä, 4th May 2015

Timo Vepsäläinen

Contents

List of Included Publications

The thesis is based on the following publications referred to as [P1] to [P8].

[P1] Vepsäläinen, T., Hästbacka, D., Kuikka, S. (2008) Tool Support for the UML Automation Profile - for Domain-Specific Software Development in Manufacturing. Proceedings of the 3rd International Conference on Software Engineering Advances. Sliema, Malta, October 26-31, 2008, pp. 43-50. DOI: 10.1109/ICSEA.2008.22

[P2] Vepsäläinen, T., Sierla, S., Peltola, J., Kuikka S. (2010) Assessing the Industrial Applicability and Adoption Potential of the AUKOTON Model Driven Control Application Engineering Approach. Proceedings of the 8th IEEE International Conference on Industrial Informatics. Osaka, Japan, July 13-17, 2010, pp. 883-889. DOI: 10.1109/INDIN.2010.5549626

[P3] Vepsäläinen, T., Kuikka, S. (2014) Integrating Model-In-the-Loop Simulations to Model-Driven Development in Industrial Control. SIMULATION: Transactions of the Society for Modeling and Simulation International. DOI: 10.1177/0037549714553229

[P4] Vepsäläinen, T., Kuikka, S. (2014) Model-Driven Development of Automation and Control Applications - Modeling and Simulation of Control Sequences. Advances in Software Engineering, Vol. 2014. DOI: 10.1155/2014/470201

[P5] Vepsäläinen, T., Kuikka, S. (2013) Benefit From Simulating Early in MDE of Industrial Control. Proceeding of the 18th IEEE International Conference on Emerging Technologies and Factory Automation. Cagliari, Italy, September 10-13, 2013, pp. 1-8. DOI: 10.1109/ETFA.2013.6647961

[P6] Vepsäläinen, T., Kuikka, S. (2011) Towards Model-Based Development of Safety-Related Control Applications. Proceeding of the 16th IEEE International Conference on Emerging Technologies and Factory Automation. Toulouse, France, September 5-9, 2011, pp. 1-9. DOI: 10.1109/ETFA.2011.6058979

[P7] Vepsäläinen, T., Kuikka, S. (2014) Design Pattern Support for Model-Driven Development. Proceedings of the 9th International Conference on Software Engineering and Applications. Vienna, Austria, August 29-31, 2014, pp. 277-286. DOI: 10.5220/0004990002770286

[P8] Vepsäläinen, T., Kuikka, S. (2014) Safety Patterns in Model-Driven Development. Proceedings of the 9th International Conference on Software Engineering Advances. Nice, France, October 12-16. 2014, pp. 233-239.

List of Abbreviations

AF	Automation Function
ASD	Automation Sequence Diagram
ASE	Automation Science and Engineering (a department at TUT)
AKM	Architecture Knowledge Management
ALM	Application Lifecycle Management
CORBA	Common Object Request Broker Architecture
DCS	Distributed Control system
DDS	Data Distribution Service
DSL	Domain Specific Language
EMF	Eclipse Modeling Framework
FB	Function Block
FMECA	Failure Mode, Effects, and Criticality Analysis
FMI	Functional Mock-up Interface
FTA	Fault Tree Analysis
GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
HiL	Hardware-in-the-loop (simulation)
HMI	Human-Machine Interface
IEC	International Electrotechnical Commission
I/O	Input/Output (transfer of data to and from application)
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
MDD	Model-Driven Development
MiL	Model-in-the-loop (simulation)
MOF	Meta Object Facility
ModelicaML	Modelica Modeling Language
OCL	Object Constraint Language
OMG	Object Management Group
PID	Proportional-Integral-Derivative
PiL	Processor-in-the-loop (simulation)
PL	Performance Level
PLC	Programmable Logic controller
P&I	Piping and Instrumentation (diagram)
QoSFT	Quality of Service and Fault Tolerance Characteristics and Mechanisms (profile)
QVT	Query/View/Transformation
SiL	Software-in-the-loop (simulation)
SIL	Safety Integrity Level
SFC	Sequential Function Chart
SysML	Systems Modeling Language
UI	User Interface
UML	Unified Modeling Language

UML AP	UML Automation Profile
TUT	Tampere University of Technology
XiL	Model/Software/Processor/Hardware-in-the-Loop (simulation)

List of Figures

1 Introduction

This Chapter introduces the topics of the thesis and provides an introduction to the background and motivation of the work. The Chapter is organized as follows. First, the background of the thesis, the research questions and the research methodology are presented. They are followed by the contributions, before outlining the organization of the thesis.

1.1 Background

Control systems are required in various applications ranging from individual and small-scale machines to extensive manufacturing plants and enterprises. The systems are required to control and supervise machines and processes in a timely and efficient manner while at the same time optimizing their productivity and guaranteeing the safety of their environment and operating and maintenance personnel. Currently, an essential role as an implementation technology of such systems is played by software control applications that are often executed on Distributed Control System (DCS), Programmable Logic Controller (PLC) or embedded platforms.

Control applications, computer programs that perform control tasks, are fairly efficient and flexible by nature. A single processing unit with a control application can control and supervise a number of complex processes, sub-processes and devices. Processing units can be connected together to control ever-larger processes while their applications exchange real-time information on the measured properties and statuses of the processes. To adapt to changing needs and specifications, the dynamic behavior of a controlled system can be flexibly altered by changing the parameters and operating points of the control application or by updating it entirely or partially. However, while the applications have become essential parts of the systems, at the same time the efficiency of their development process has become an essential competitiveness factor in the domain.

Development of a control system for an industrial plant, for example, is a complex endeavour. It requires the collaboration of experts and information from various domains of expertise. Control system development, and control application development as part of it, requires and integrates information from process, electric, hydraulics, safety and chemical engineering, for example. Some of these engineering disciplines may also require information from control application development. However, in a common case it is the control application that can and need to adapt to

requirements and conditions from the other disciplines. To cope with the amount of information and requirements, the use of models – to complement or to substitute written documents - has been studied in the domain. However, models and modeling concepts alone are not the answer. They need appropriate, flexible tool support for performing the required engineering activities within a model-driven development process.

Modeling concepts developed for the needs of automation and control domain need to be supported by a modeling tool, including their possible relations to the concepts of more general purpose modeling languages: UML and SysML. UML and SysML based modeling techniques that have been widely used in MDD are a sound alternative for also control applications and enable modeling from the early stages of development. The models need to be processed with model transformations to automate repetitive but error-prone tasks and in order to streamline information transfer from and to the related engineering disciplines. Especially at the early stages of adopting MDD technologies to practical use, modeling concepts need to be implemented in a flexible manner for future needs. Re-use of existing knowledge and design information has to be supported in models in order to obtain the benefits of re-using application blocks, which is already reality in control application development.

Using models as primary artefacts during development offers possibilities that are beyond the capabilities of current control system and application development practices. Models that are formal enough can be analyzed and studied either alone or together with the models of the processes to be controlled. In control algorithm design, simulation is a technique that has been traditionally used to study and experiment possible control approaches, structures and tunings. However, traditionally the activity has been separated from the basic control application development. Simulation studies have been possible only after developing the applications, by executing them in conjunction to the models of processes to be controlled.

If models are to be used as the primary engineering artefacts, they should also serve documentation purposes for which information is currently produced mainly with manual work. Safety related systems, especially, constitute an area of applications in which documentation is of special importance because of the need to be able to prove the compliance of the produced applications to standards and to convince the authority of the correctness of the application. However, it is also an area of applications that could especially benefit from the use of models. Models could enable automated

consistency checks during design and transferring the design information to a form in which it answers the relevant questions.

The motivation of the thesis is to study how control application engineering could be facilitated by extending a Model-Driven Development (MDD) approach. The thesis focuses on concepts and tool support for modeling, model processing, integrated simulations and safety-related information in models.

1.2 Research Questions

The thesis explores whether or not the control application development can benefit from MDD and selected technologies enabled by it. To answer this general question, the thesis focuses on a set of smaller research topics. They are related to modeling and developing tool support for modeling the applications, ability to integrate and gain benefit from integrating simulations into MDD and ability to document safety-related information on control applications in models. These research topics are divided into three groups of questions hereafter referred as RQ1-3.

1. How to develop support for domain-specific, UML based modeling in control application development? How to develop support for and gain benefit from applying design patterns in models? How to enable and gain benefit from re-using platform specific blocks in modeling?
2. How can model-in-the-loop simulations be integrated into MDD of automation and control applications with UML based modeling? What are the requirements and constraints for selecting the simulation approach to be followed? How can simulations with the selected approach benefit MDD?
3. How can the safety of control applications be supported in MDD? How can risk and hazard information be integrated into modeling? How can traceability, correctness and completeness be supported in models? How can the use of design patterns support documenting the safety features of control applications?

1.3 Scope of the Thesis

The thesis discusses the use of MDD and techniques enabled by it in automation and control application development. The main focus of the thesis is on whether and how control application development could be enhanced with MDD techniques and how the required tool support can be implemented with the use of standard techniques.

Related to implementing the domain specific modeling concepts and tool support for a MDD process, the thesis studies and uses standardized modeling, metamodeling and model processing techniques. Graphical support for the modeling concepts, which is in current modeling tools often implemented on top of the information content layer, is not considered in detail in the thesis.

In the thesis, simulation is considered as a means to evaluate and compare control application designs. It is also a technique that is already in use in the domain; however, not necessarily during basic control application development. Simulations are widely used in control algorithm development and in, for example, control system testing after the development. In the thesis, the use of simulations as well as techniques and approaches to create closed-loop simulations are discussed with focus on the software development phase. However, especially related to interlock functions the distinction between algorithm and software development is sometimes difficult to make.

Related to safety aspects and safety related information in models, the thesis focuses on extending the information content of models with IEC 61508 [1] as a reference. The purpose is to develop the MDD process and concepts in a direction in which they could fulfill more of the requirements of the safety standard. However, with discussion on documentation, the author does not want to claim that safety systems should or should not be developed with MDD techniques only. Nevertheless, in order to develop safety systems with MDD techniques, it would be vital to be able to fulfill the relevant documentation requirements with MDD.

1.4 Research Methodology

The research methodology of the thesis is the constructive approach of design science. According to Iivari [2], design science research has been applied in computer science, software engineering and information systems for decades producing e.g. new architectures, languages and algorithms. It is the rigor of constructing IT artifacts that distinguishes the design science from the practices of building IT artifacts and to demarcate the two there are two options. The essence of information systems can lie in the scientific evaluation of the artifacts or in a reasonable rigorous constructive research method for building the artifacts. [2]

According to Crnkovic, the key idea of constructive research is the construction based on the use of existing knowledge in novel ways and possibly adding new links. The construction proceeds through design thinking to the projections of future solutions. Conceptual and other knowledge gaps are filled with purposefully tailored building

blocks to support the whole construction. When a construction, theoretical or practical, differs profoundly from pre-existing ones, it constitutes a new reality against which pre-existing ones can be examined and understood [3].

According to Hevner and March [4], the purpose in design science is to create innovations or artifacts that embody ideas, practices, capabilities, and products that are required to efficiently accomplish the analysis, design, implementation and use of information systems. According to [4], the output artifacts of research include constructs, models, methods and instantiations. However, due to the range of output research artefacts in reported research, a more expansive view of the artifacts can include any designed solution that solves a problem in a given context [5].

The main research steps applied in the research are as follows:

1. Tool support for the domain specific modeling concepts of UML Automation Profile is developed while taking into account the needs of the application domain related to the re-use of application blocks, for example. The industrial applicability of the model-driven development process and tools as a part of it are assessed.
2. The use of design-time, closed-loop simulations is investigated to facilitate control application development. Methods are developed for generating simulation models from UML Automation Profile models and to integrate the models in a novel manner to plant simulation models. General approaches to closed-loop simulations in MDD in the domain are compared.
3. The use of design patterns is studied to enhance the re-use of existing design solutions. Modeling concepts and tool support are developed for specifying design patterns, marking and visualizing design pattern instances, applying design patterns as well as for using patterns to produce documentation from models.
4. The modeling concepts are extended to enable the specification of how the hazards associated with the controlled systems may occur. Traceability, correctness and completeness are improved within models with safety aspects in mind.

The evaluation of the results is performed in each step with respect to the fulfillment of requirements, comparison to the state-of-the-art as well as evaluation of improvements in comparison to the state-of-the-art.

1.5 Contributions of the Thesis

The scientific contribution of the thesis and included publications to the research questions are following.

RQ1 Modeling and model processing

- Development of a modeling tool, by extending existing open source modeling tools and frameworks, so that modeling concepts are implemented on metamodel level. The models are extendable and they can be processed with standard model transformation languages.

- Qualitative assessment of industrial applicability of the experimental MDD development process and tools as a part of it.

- Development of an approach to specify design patterns and mark design pattern instances in UML, SysML and UML Automation Profile (UML AP) models to support the learning of developers, the traceability of solutions and producing documentation from models.

- Development of a method and tool support for modeling control application implementation block libraries with stereotypes and model libraries so that code generation can utilize existing blocks within produced executables.

RQ2 Simulation of design models

- Development of an approach to transform UML AP control application models to existing Modelica Modeling Language plant simulation models to enable design-time closed-loop simulations.

- Extending the simulation approach to cover the aspects of basic control systems including feedback control, binary control, sequential control as well as interlocks.

- Comparing the transformation assisted approaches to closed-loop simulations in MDD in control application development. Assessment of benefits of design-time simulations based on case studies and literature.

RQ3 Modeling safety features

- Specification of modeling concepts to describe how hazards, related to a controlled system can occur, to extend the documentation value of the models and to support the traceability of requirements.

- Development of documentation exporters to support correctness, completeness and traceability in MDD and in models.

- Extending the design pattern modeling concepts for the patterns of safety systems and developing methods to use the concepts in generating safety documentation from models and guiding development work.

1.6 Organization of the Thesis

The research questions, RQ1-3, are addressed in separate Chapters of the thesis and in the included publications as indicated in Table 1. The publications appear in the table in the order of their importance related to the research questions.

Table 1 the included publications, the research questions and their presentation in the thesis.

Research Question	Publications	Thesis Chapter
1 Modeling and model processing	P1, P2, P7	3
2 Simulation of design models	P3, P5, P4	4
3 Modeling safety features	P6, P8, P7	5

Chapter 2 introduces the technologies and standards that have been applied in the thesis and in the included publications. The general purpose technologies are in the thesis applied to model-driven development of control applications. The standard introduced in Chapter 2 is a functional safety standard that has been used as a normative reference when extending the information content of models to safety aspects. The means to support domain-specific modeling with focus on the re-use of both design patterns and concrete implementation blocks and automated model processing are discussed in Chapter 3. Simulations and techniques to enable and benefit from simulations of design-time models in model-driven development are discussed in Chapter 4. Extensions to modeling concepts and model processing tools for supporting safety related information in models are presented and discussed in Chapter 5. Chapter 6 provides a summary of the included publications. Chapter 7 concludes the thesis with the re-examination of the research questions and an outlook of future research.

2 Technological Background

The focus of the thesis is in model-driven development (MDD) of control applications. Control applications are software parts of control systems which perform control tasks. In industry, the control applications are typically executed in Distributed Control System (DCS), Programmable Logic Controller (PLC) or embedded platforms. They are used in the real-time control of processes of various kinds ranging from mobile working machines and platforms to, for example, chemical industry and power production plants. In addition to real-time control, the control applications of this kind can include, among others, monitoring and safety features. However, safety critical control functions the sole purpose of which is to guarantee the safety of the processes to be controlled are typically implemented in dedicated safety systems. The role of (basic) control applications, on the other hand, is to keep the processes in their normal, profitable operation regions. The focus of the thesis is in the basic control systems.

Following is a brief introduction to the technologies and methods that are used in the thesis and included publications to enable or to facilitate the development of control applications with the use of models.

2.1 Modeling and Model-Driven Development

2.1.1 Model-Driven Development

Model-Driven Development (MDD) is a software development methodology that emphasizes the use of models as primary engineering artefacts during the development of applications. Acronyms related to MDD include, among others, model-driven engineering (MDE) and model-driven architecture (MDA) [6], the latter being a registered trademark of Object Management Group (OMG). In MDD, models of different phases and accuracy levels are used to contain the information about the system (application) during the development of it. The models, starting from, for example, requirement models, are developed, elaborated and refined with automated model transformations and manual work. The role of the transformations is often in automating the creation of later phase models based on former ones. In software development, the goal of the development process is often an executable application, which (or part of which) can also be possible to be produced automatically with one type of model transformations, with code generation.

2.1.2 UML and SysML

Unified Modeling Language (UML) is a software modeling language that defines both the information content of modeling elements and the graphical notation of diagrams conforming to the language. The first official version of the language was adopted by OMG in 1997 [7]. A major improvement to the language was version 2.0 that included improvements and clarifications to the metamodel and semantics of the language [8]. The metamodeling technique used to specify the UML metamodel is Meta Object Facility (MOF) [9], which has also been specified by OMG. The current officially adopted version of UML is 2.4.1 [10], [11].

UML is currently the de-facto modeling language for the modeling of software systems and applications including their requirements, structure and behavior. The modeling concepts of the language are closely related to concepts in object-oriented programming languages. However, the language can and has been used to describe the aspects of e.g. procedural PLC applications. UML has been designed to be extendable for special purposes and needs of specific applications domains. For example, SysML [12] has been developed for systems engineering purposes with the use of the profile mechanism of UML. The mechanism utilizes stereotypes to alter the semantics of the elements. In addition to the profile mechanism, an alternative to extend UML is to apply metamodeling, by extending the modeling elements of the language with the metamodeling technique (MOF) that has been used to specify them in the first place.

Systems Modeling Language (SysML) [12] is another graphical modeling language specified by OMG, for systems engineering purposes. The language has been defined as an extension to UML, by re-using parts of UML (UML4SysML), altering parts of UML and adding new modeling concepts and diagrams. Whereas UML is software centric, SysML is less restrictive related to the implementation of the models. Blocks of the language, which correspond to UML classes, are suitable for representing hardware blocks and parts of systems, for example.

2.1.3 Metamodels and Meta Object Facility

The modeling concepts that can be used in models conforming to a modeling language are defined in the metamodel of the language. Metamodels, thus, define the concepts available for modeling including their properties and other information content as well as relations to other concepts. In addition to defining modeling languages, metamodels can be used when defining model transformations between languages.

Models conforming to a modeling language are instances of the metamodel of the language similarly to metamodels being instances of metametamodels, which can be used to define metamodeling languages. The metamodel of UML, for instance, has been defined with MOF. MOF, on the other hand, defines itself so that a metametamodeling language has not been needed for defining MOF. With respect to metamodeling layers, real world objects can be described being level M0 and instances of model elements on level M1. Models on level M1 are instances of metamodels on level M2 whereas metamodels on level M2 are instances of metametamodels on level M3.

2.1.4 Model Transformations and QVT

Model transformations are processes that are used to ensure, by modifying one or more models, that the models processed by the transformations are consistent with each other. Model transformations can be further divided into model-to-model transformations, which are used between models, and model-to-text transformations, which are used to create text (e.g. code) based on models. Model-to-model transformations are thus processes that create or update models or parts of models based on the same or other models or parts of them. Model transformations can be performed automatically, by a computer program, or manually with operations that are manually performed by a modeler. In model-driven development, a common goal is to automate model transformations that are repetitive, which reduces the amount of required manual work and potential for errors.

Query/View/Transformation (QVT) [13] is a model-to-model [14] transformation language that has been specified by OMG for defining transformations between models that conform to modeling languages that have been defined with MOF. The language specification defines three distinct languages: Core, Relations and Operational Mappings. By nature, Core and Relations languages are declarative whereas Operational Mappings language is imperative. With respect to the metamodeling layers, QVT language can be regarded to be on layer M2, similarly to UML metamodel, for example. Individual model transformation (specification) instances are on level M1 and utilize the concepts of the source and target metamodels on layer M2. Executable model transformations, which are instances of their specifications, manipulate models and modeling elements on layer M1. The metamodeling layers as well as relationships between metamodels, models, transformation definitions and model transformations are illustrated in Figure 1 that has been modified from [15].

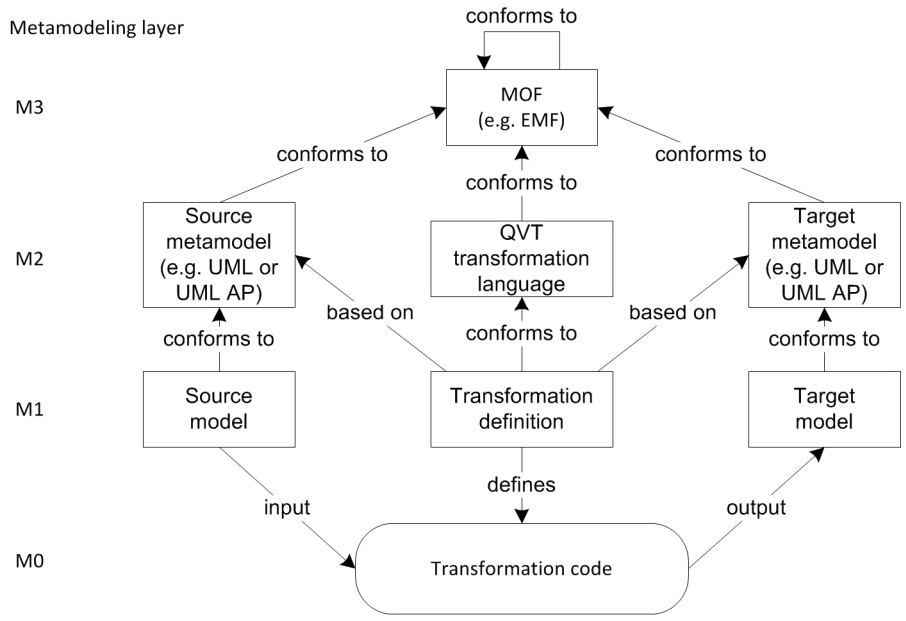

Figure 1 The relationships between metamodels, transformation definitions, models and model transformations.

2.2 Simulations

2.2.1 Overview

Computer simulation is a technique that can be used to imitate the operation of a process or system based on a model of the process or system in order to predict, study or explain the behavior of it. In control system and application development, simulations can be used e.g. in the design and validation of control programs, strategies and human-machine interfaces before installing the complete systems [16]. In control application development, a closed-loop simulation requires a simulation model of the system to be controlled and a component acting as the control system in the simulation.

A closed-loop simulation can be executed within a single simulation engine or as a co-operative simulation (co-simulation). In the latter approach, two or more simulation engines are connected together and execute the parts of the simulation model. For example, the parts can be a simulation model of the system to be controlled and the model of the control system and/or application controlling the former one.

2.2.2 XiL Simulations

XiL simulations refer to the 4 simulation approaches that can be used in conjunction to model-based development: model-in-the-loop (MiL), software-in-the-loop (SiL),

processor-in-the-loop (PiL) and hardware-in-the-loop (HiL) simulation [17]. In MDD of control applications, these approaches differ in the control system configurations used to control the simulation model of the process to be controlled.

In MiL, a model of the control system and or application is used whereas SiL, PiL and HiL utilize software generated from the model, generated software with its target processor and generated software with the entire target control system hardware, respectively. Similar simulation approaches, except MiL, can be used to test control applications in more conventional application development processes. HiL simulation, for example, can be used to test a control application with its target control system hardware regardless of the process to develop the application.

2.2.3 Modelica and ModelicaML

Modelica is a non-proprietary, object-oriented, acausal language for the modeling of heterogeneous physical systems [18], [19]. It supports the use of libraries and multi-domain modeling so that the modeled systems may include, among others, mechanical, electrical and control subsystems. Modelica models are mathematically described with differential, algebraic and discrete equations [19]. Modelica models can be defined both textually and graphically, depending also on tool support.

ModelicaML is a UML profile that has been developed to enable creating, reading, understanding and maintaining Modelica models with UML tools. [20] The profile uses a subset of UML concepts and defines a set of stereotypes, with stereotype specific tagged values, that are given semantics by the Modelica language. The profile has been implemented on Eclipse platform based on UML2 implementation of the UML metamodel. The profile is currently tool supported so that ModelicaML models can be transformed to textual Modelica code and simulated with a Modelica tool [21].

2.3 Safety

2.3.1 Overview

Safety can be defined as freedom from an unacceptable risk. The risk concept can be defined as a combination of the probability of occurrence of harm and the severity of the harm [1]. Functional safety is part of the overall safety relating to the system of interest (equipment under control and its control system) that depends on the correct functioning of the electrical/electronic/programmable electronic safety-related systems and other risk reduction measures. [1]

A practical definition for software safety, provided in [22] is: features and procedures that ensure that a product performs predictably under normal and abnormal conditions. The likelihood of an unplanned event occurring is minimized and its consequences controlled and maintained; thereby preventing accidental injury or death, whether intentional or unintentional [22]. In the automation and control domain, the safety of the controlled plants, processes and machines often needs to be ensured by functional safety systems that perform safety functions and include software parts.

2.3.2 IEC 61508

IEC 61508 [1] is an essential standard in the domain of functional safety. The standard has been renewed a short while ago, in 2010, so that with respect to its recommendations the standard is still as modern as applicable. The standard is a basis for several sector specific standards, e.g. IEC 62061 in machinery [23] and IEC 61513 for nuclear power plants [24], which increases its importance. IEC 61508 covers the functional safety of systems containing electrical, electronic and/or programmable electronic systems. Software applications as parts of the programmable electronic systems are covered in the third part of the standard. The standard defines an overall lifecycle model for safety functions, according to which they can be specified, developed and maintained.

The standard has been built so that a natural way to fulfil the requirements of it would be to utilize the traditional V-model development process. However, provided that the requirements are fulfilled, any development process can be used [P6]. Safety functions that consist of electrical parts, for instance, are treated by the standard based on the probabilities of correct operation. However, because of the systematic nature of software faults, in case of software safety functions the standard focuses on software development techniques and measures. It guides their selection as well as the information content of documentation that must be produced to develop certifiable applications to safety systems. In the thesis, the standard and the requirements of it are used as a basis for extending the information content of models of basic control systems with safety aspects and features.

2.3.3 Systematic Safety System Development and Patterns

Generally, the concepts of safety and reliability are well understood in relation to, for example, electronic components. However, software safety and reliability form a discipline that is well understood by few [22]. Unlike hardware, software does not break, fail or wear out over time. The causes of software failures are systematic, not

random [22]. Because of the enormous state spaces of digital systems, it is also possible that only a small part of causes of the failures can be exercised with testing [25].

System safety, in contrast, integrates management, hazard analysis and design approaches to a planned, disciplined and systematic approach to prevent or reduce accidents throughout the system lifecycle. System safety attempts to predict accidents before they occur and to eliminate or prevent hazardous states. The primary concern in system safety is, thus, the management of hazards in a controlled and systematic manner. [25]

In software engineering, design patterns are a means to systematically re-use well-known, proven solutions. Each design pattern systematically names, motivates and explains a design solution that addresses a recurring problem or challenge in system designs [26]. For safety systems, suitable design patterns can be found from both standards and related literature. For example, IEC 61508 (in the third part) lists architectural approaches and solutions, many of which have been presented in a more detailed manner in pattern literature. For example, the standards suggest the use of redundancy [27], backward recovery from faults [28], [29] as well as cyclic program execution [27].

3 Tool Support for Model-Driven Development of Control Applications

The use of models and model-driven development techniques has drawn extensive research attention in the domain of automation and control systems during the past few years. The modeling of software applications and systems, including their requirements, has been seen as an integral phase in development and as a means to cope with the increasing size and complexity of the applications. Such work has been published related to both IEC 61131-3 [30] and IEC 61499 [31] based control system platforms. Of these languages, IEC 61131-3 is a standard that defines five PLC programming languages. The languages include Function Block (FB) diagram, structured text, sequential function chart, ladder diagram and instruction list. IEC 61499, on the other hand, extends the FB concept of IEC 61131-3 with event-driven execution and support for distributing FBs in de-centralized execution environments.

With IEC 61499 [31] as a target language, Thramboulidis and Tranoris have studied and developed tools [32] and an engineering process [33] for distributed control applications using UML to present requirements and design before implementations. The approach of the EU MEDEIA project [34] builds on the use of Automation Components and bi-directional model transformations between models. Automation Components are described as composable combinations of embedded software and hardware. Vyatkin et al. [35] have developed a model-integrated design framework for automation and control applications that is based on an intelligent mechatronic component concept and use of the IEC 61499 architecture. Of the referred approaches, [34] and [35] discuss also how design models could be simulated, which is the topic of Chapter 4 of the thesis. Other approaches related to combining the use of IEC 61499 and UML in the domain include the work of Dubinin et al. [36], Hussain and Frey [37] as well as Panjaitan and Frey [38].

Related to IEC 61131-3 [30] as a target language, FLEXICON project, see [39] and [40], has integrated a combination of commercial off-the-shelf tools for supporting software development of both basic control and safety related control systems. MAGICS approach [41] aims at non-device-centric abstractions and support for PLC (IEC 61131-3) code generation that is claimed to be missing from many approaches. The approach ([41]) also addresses sequential control activities. Related to generating PLC code from models, mappings between UML and IEC 61131-3 as well as the earlier

version of it have been presented by Witsch and Vogel-Heuser [42] as well as by Vogel-Heuser and Witsch [43].

Use of design patterns in the domain has not been addressed in many MDD approaches. However, Witsch and Vogel-Heuser in [42] envision collecting known solutions to pattern catalogues in order to improve their re-use, motivated by the object-oriented extensions to IEC 61131-3. For example, implementing a structure such as the one in Observer design pattern [26] requires object-oriented features of programming languages. In application domains other than industrial control, techniques and support related to design patterns have been developed to specify patterns [44], to apply and evolve patterns to models [45], [46], to detect pattern instances [47], to detect points in models where patterns could be applied [48] as well as to visualize pattern instances in models and diagrams [49], [50].

This Chapter discusses the development of tool support for domain specific modeling and MDD in automation and control domain. The AUKOTON MDD process, which is to be supported, is introduced briefly in Section 3.1. Domain requirements for the tool are presented in Section 3.2. Possible implementation techniques are discussed in Section 3.3. Section 3.4, then, introduces the developed UML AP tool and discusses choices related to the development of it.

3.1 AUKOTON Development Process

AUKOTON is a development process for automation and control applications that was developed during AUKOTON project. In detail, the process has been presented in [51]. However, it has been discussed also in the included publications [P1] and [P2]. The development process aims to apply model-driven development technologies to control application development while at the same time taking into account domain specific practices related to, for example, the re-use of existing implementation blocks. The process emphasizes the importance of platform independent modeling, automated transfer of design information and late binding of platform specific details. The objectives are to enhance productivity, solution re-use and software quality [P1].

The modeling basis in the process is UML AP [52] that covers the essential concepts of modern, complex automation applications. The profile was further developed during the project with respect to both requirement modeling and functional modeling concepts. The development process, see Figure 2, applies models in three phases. The names of the phases are *requirement import and elaboration*; *functional, platform independent design;* and *functional, platform specific design*.

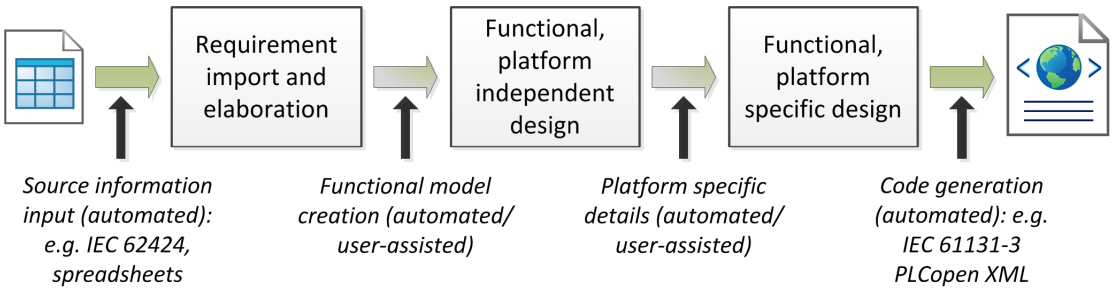

Figure 2 The AUKOTON development process proceeds from requirements to executable applications through the requirement, functional and platform specific development phases.

During the requirements phase, UML AP requirement concepts are used to describe the required functionality as well as non-functional properties of the applications. Part of the information can be imported to the phase from source information documents e.g. IEC 62424 [53] Piping and Instrumentation (P&I) diagrams or MS Excel spreadsheets that can be produced e.g. by the process and instrumentation design. Spreadsheets, with company specific practices, are also commonly used in industry [P2]. Such documents can contain vital information about required control functions as well as connection points in the processes to be controlled for controls and measurements. The Imported requirements, as well as other intermediate products of the development process, can be inspected and refined by developers in order to add information and decisions that are not automated. The requirements are described mainly with the structured Automation Requirement concepts of the profile but also informal textual requirements can be used.

During the functional, platform independent design phase, the functionality of the applications is specified in a platform independent manner but so that it can be later refined with platform specific details [P2]. The purpose is to increase the re-use potential of models so that design work could be re-used also in projects that are targeted to other control system platforms. During the phase, the modeling concepts of interest are the Automation Function (AF) concepts of the profile (UML AP).

The central AF concept has been further divided into a hierarchy of different kinds of measurements, actuation, control and interlock functions. AFs represent individual pieces of the applications. They could be characterized as platform independent, abstract type circuits (function blocks) representing different kinds of measurement, actuation, control and interlock functions that can be combined and connected together to compose an application [P2]. However, for each AF, there can exist several concrete type circuits - possibly on different platforms - that could be used to implement the functionality. That is, AFs neither identify the type circuits to be used nor restrict the selection of the target platform. AFs exchange information with Ports that specify both

their types and roles, from the point of view of the AFs. For example, it is possible to define a Port to be intended for relaying measurement information.

During the functional, platform specific phase, the purpose is to detail the platform independent design for a chosen platform so that the application code can be generated [P2]. Appropriate AFs (of the platform independent model) are tied to platform specific implementation blocks to be used in the final applications. The connection interfaces of the AFs are completed to correspond to those of the blocks and the required parameters of the blocks are set. For example, for the assessment of the development process and tools [P2], a set of type circuits that had been developed as IEC 61131-3 FBs was modeled as an AUKOTON DCS collection. The collection was used for the generation of an executable application in PLCopen IEC 61131-3 XML format [P2].

In the development process, see [51] and [P2], model transformations are used between source information documents and requirement models, between requirements models and functional platform independent model and between platform specific models and executables. Between platform independent and platform specific modeling phases, the process uses an interactive model transformation that reads and modifies a single model. Thus, three types of transformations are required by the process: 1) import transformations that import information to a model or a model Package, 2) intra-model transformations that read and modify Packages of a model and 3) export transformations that produce e.g. documentation files or parts of executables based on models or Packages. All the transformations are automatic; however, after executing transformations the resulting models can be edited manually. The transformations produce rather starting points for manual work than complete phase products of the development phases.

A single modeling tool is used throughout the AUKOTON process. The tool shall support the entire application development process starting from manufacturing oriented requirements and proceeding via platform independent design to platform specific implementations. During the process, the tool must enable the use of the required diagram types and concepts of UML AP. During the requirement import and elaboration, the process utilizes mainly Requirements Specification Diagram. During the functional modeling phases, the diagram types of interests are Control Structure Diagram and Automation Sequence Diagram, both of which may not always be required, depending on the modeled application.

3.2 Requirements for Modeling and Model Processing Support in MDD of Control Applications

3.2.1 Modeling Concepts and implementations

The use of the AUKOTON development process in control application development, with domain specific modeling concepts, requires tool support for the entire application development lifecycle. Support shall start from source information and requirements and proceed, via platform independent, architectural considerations, to the platform specific implementation. [P1] Development of a MDD tool with consideration for domain requirements and practices was, thus, an important research task from the beginning of this research. Tool support was also required to further improve and to experimentally estimate the profile (UML AP) that had been previously specified [P1]. The development work begun in the AUKOTON project, during which UML AP was initially applied to MDD. Thereafter, tool development has been an on-going activity during which both the modeling concepts and techniques to benefit from models have been further developed.

Applying MDD techniques also requires taking into account various application domain specific and other requirements and characteristics. These requirements are briefly discussed in this and following sub-sections related to the modeling concepts and their implementation techniques, development of graphical tool support, use of model transformations as well as re-use of design patterns and concrete implementation blocks.

UML AP, in its initial form [52], was partially extended from the suitable concepts of SysML [12], the UML Profile for Schedulability, Performance and Time [54] as well as UML Profile for Quality of Service and Fault Tolerance [55]. Although the profile defines new diagram types, not all the concepts of it are intended for them [P1]. Instead, many of the extended concepts are intended for UML and SysML diagram types in which they can be used as stereotypes, so that practical use of parts of the profile requires support for UML and SysML. It was, thus, a clear requirement that the profile implementation should be based on an existing UML/SysML tool to enable the co-use of the languages without developing UML and SysML support from scratch [P1]. However, significant parts of the concepts of the profile are new, specific to the domain and intended to be used in new, domain specific diagram types. (The new diagram types are intended to describe the requirements of control applications, control structures as well as sequentially executed control activities.)

In addition to be used with UML, the UML AP modeling concepts were required to be extendable and flexible for future needs; so that the profile and concepts could be further developed [P1]. Modeling languages undergo major changes infrequently so that their implementations do not need to be updated every day. Changing the metamodel of a tool does not need to be as easy as modifying a graphical application model. However, changes need to be realizable with a reasonable amount of work. UML AP has also been further developed during the research to enable e.g. modeling control logic and hazards. The fulfilment of the extendibility requirement has thus been evaluated during the research.

3.2.2 Graphical Support

Implementing UML AP required implementing the new graphical diagram types. The new diagrams resemble domain specific diagram types and notations and are thus intuitive for domain professionals. For example, the Automation Sequence Diagram type is based on the Sequential Function Chart (SFC) notation which is part of IEC 61131-3 [30]. The purpose of the (intended) resemblance is to make it easier for domain professionals to familiarize themselves with UML based modeling and tools. With a Domain Specific Language (DSL), problems can be solved with domain concepts, on a high level of abstraction and in a problem-oriented manner. However, as a drawback, design and implementation of a DSL require a lot of effort and consideration [P1].

Graphical support development requires the stability of modeling concepts. On Eclipse platform, with existing open source tools, the supported approach to build graphical modeling support is to develop diagram types on top of a model layer, in a layered like architecture. In this way, graphical code manipulates models that are on a lower level. Graphical diagrams need the information content of models and their metamodel level properties. As a consequence, code related to implementing graphics often requires changes when the model code (metamodel) is changed[1]. Changes, however, are not required on a daily basis and it should not be possible to cause changes to metamodel e.g. by accident.

[1] Graphical tool support development is in this thesis addressed only to the extent to which graphical tool development is affected by choices in modeling concept implementations.

3.2.3 Model Transformations

In MDD, model transformations are the means to reduce the amount of manual development work and to automate tasks that are repetitive enough to be treated with programmed rules. Transformations can be used in importing information to models [P1, P2], transferring information between modeling and development phases [P2], generating code [P1, P2] and generating documentation from models [P6, P7, P8]. Transformations and related techniques, e.g. QVT and Object Constraint Language (OCL) [56], can be used to query models and to automate consistency checks [P6]. It is also possible to use model transformations for creating simulation models that can be used to assess designs in a timely manner [P3, P4, P5].

The models that are used in a MDD process thus need to be processable with (preferably standard) transformation techniques. OMG, for example, has specified three QVT [13] model transformation languages for which there are open source implementations on Eclipse platform, e.g. SmartQVT[2]. Transformations should be integrated into the MDD environments so that all transformations could be searched and controlled in an agreed manner, with a graphical user interface of the tool. However, similarly to graphical modeling support, model transformations often need to undergo changes when modeling concepts change. On the other hand, to support e.g. new source information formats or control system platforms, the integration must be loose and adaptable [P1]. It must be possible to add, remove and replace transformations in a flexible manner. It cannot be assumed that all transformations that may be required in future would be known or would have been known.

Transformations, thus, differ from each other by their basic purpose. However, they also accept different parameters, which must be taken into account in the development of the transformation support mechanism [P1], [15]. For example, in the AUKOTON process, code generators usually only read source models whereas intra-model transformations both read and modify parts of a model. On the other hand, to import information to a requirement Package from several sources, for example, it must be possible to target transformations to the selected Packages of a model. [15]

[2] http://sourceforge.net/projects/smartqvt/

3.2.4 Design Patterns

The efficiency of control application development work is becoming a more and more important competitiveness factor in the domain. A means to improve the efficiency of the work is facilitating the re-use of work and solutions. In the domain, such solutions to re-use can be concrete, platform specific implementations and blocks, the re-use of which is already common. However, re-use should be supported also with respect to general, platform independent solutions and structures. Due to the lack of acknowledged methods for supporting the platform independent development [57], for instance, their re-use has not been as common as in the case of platform specific design.

Design patterns, see [58] and [59], are a means to re-use platform independent solutions. A design pattern represents a relation between a context, a problem and a solution [59]. Patterns document proven solutions to recurring challenges in design and development work and capture expert knowledge for re-use purposes, for both expert developers and less experienced ones [P7]. In the domain, an example of a design pattern could be organizing a measurement, a controller and an output to a control loop.

Patterns have names that are known to developers so that their use aids communication. They provide vocabulary for developers, enhance documentation and encapsulate knowledge and experience. [60] A design pattern instance marks a point in which a developer has been potentially faced with a challenge (that the pattern addresses). Pattern instances represent design decisions to use patterns, with pattern specific potential benefits and drawbacks. The use of patterns could thus be of great value and extend the documentation value of models towards architectural knowledge. Especially this could be useful in MDD that emphasizes the use of models instead of (written) documents. If documents are not used in a development process, the only places where the information can be added are the models [P7].

To benefit from patterns, a non-restrictive pattern modeling approach is required. UML, as the de-facto software modeling language, aims to support patterns with its Collaboration concepts. However, as presented in [P7], a pattern modeling approach should not restrict the nature of solutions in patterns. Patterns should be able to consist of any modeling elements such as class definitions or components, not only the properties of UML Classifiers, as is the case in the UML approach [P7]. On the other hand, it should be possible for other modeling elements than Classifiers to contain elements that play roles in pattern instances [P7]. To systematically use and benefit from the use of patterns, it should also be possible to collect patterns to libraries as suggested e.g. in [42].

Automating the application of patterns to models could be a useful feature. However, even without it, patterns could be used to document recurring solutions and their use for e.g. documentation and traceability purposes. Pattern concepts should also enable generating traceability information and statistics on their use. It should be possible to visualize patterns in models and diagrams so that they could improve the documentation value of the diagrams and learning of developers. With an appropriate tool support, patterns could also enable comparing applications in terms of re-use as was done e.g. in [61] with respect to the re-use of platform specific engineering work. [P7]

3.2.5 Platform Specific Implementations

In addition to platform independent models and solutions, re-use can be related to platform specific blocks. The re-use of implementation blocks, e.g. type circuits that perform control algorithms or interface with the sensors and actuators, is a special characteristic of the domain. As such, it needs to be taken into account when developing tool support for the AUKOTON development process. As argued in [57], DCS platforms capture the results of years of development and well-tested features that are worth supporting. Ability to re-use existing, tested and known blocks could increase quality and reduce the amount of repeated work also within MDD.

To enable the re-use of implementation blocks, it should be possible to refine platform independent design to platform specific design. In the AUKOTON process, parts of platform independent models need to be possible to be refined to platform specific ones that are then used in executables. In order to use code generation to produce applications, the required information should be available in models. That is, it should be possible to use the platform specific features of implementation blocks, e.g. interlock ports, and it should be possible to set platform specific properties in models.

3.3 Considerations on Implementation Techniques

3.3.1 Extension Mechanisms of UML and MOF Based Languages

UML can be extended with two distinct approaches: by using the built-in, stereotype based profile mechanism of it and by extending the metamodel of the language with the use of Meta Object Facility (MOF) [9]. MOF is the metamodeling technique that has been used in the first place to define the metamodel of UML. These two approaches were also the practical alternatives for implementing the UML AP modeling concepts

[P1]. However, the mechanisms differ in terms of modifications that they enable and in terms of required work.

With the built-in (light-weight) mechanism, extensions are defined as Stereotypes that can be used to specialize the semantics of the modeling concepts of the language. Stereotypes can also define tagged values, which are attributes with basic data types. The tagged values can parameterize the semantic characteristics of the Stereotypes. Stereotypes, however, cannot be used in a way that would contradict with the UML metamodel [10]. For example, the use of Stereotypes to insert new metaclasses or meta-associations between metaclasses is prohibited. This is a clear restriction of the approach, since some of the concepts required by the new UML AP diagram types have structural features that do not fit the UML metamodel. Implementing these concepts requires at least new meta-associations, in addition to defining Stereotypes [P1]. In the MOF-based approach, there are no such limitations related to the addition of new elements [P1], [10]. Removing existing metamodel elements from an extended tool, however, could be difficult if the concepts were implemented with program code in an extended tool.

Both the extensions mechanisms are, to some extent, tool-supported. The Stereotype based mechanism, for example, is supported by standard tools such as Magicdraw[3] and Topcased[4] so that no programming work is required. Stereotypes can be defined in profile models that are referenced by application models in which the Stereotypes are used. In this way, models with domain specific extensions can be portable to other tools (with compatible file formats). However, UML profile models cannot define new graphical diagram types in typical tools. (Although new diagram types are sometimes described in written profile specifications such as that of SysML [12].) As a consequence, to support new diagram types, programming work is often required in any case. On the other hand, with special diagram types, models may not be portable to other tools regardless of the implementation technique of the modeling concepts.

The metamodeling based approach often requires additional programming work (compared with the Stereotype based approach) since modifications to the metamodel require changes in program code. For example, new metaclasses usually require implementing code for them so that the new code is coupled to implementations of

[3] http://www.nomagic.com/products/magicdraw.html

[4] http://www.topcased.org/

existing metaclasses. Metamodel modifications can also affect adversely on the portability of models. It is possible that models containing instances of new metaclasses cannot be opened in other (standard) tools. However, as mentioned, with new diagram types this can be the case regardless of the modeling concept implementation technique. This is because the models would include information related to the new diagram types and elements in them.

3.3.2 Graphical Diagram Development on Eclipse Platform

At the time of AUKOTON project and beginning of the tool development, there were at least two alternative tool families that supported graphical tool development. These alternatives were: 1) the use of Graphical Editing Framework (GEF[5]) and Graphical Modeling Framework (GMF[6]) of Eclipse Modeling Project[7] and 2) the use of Topcased as the extended base tool. Both the alternatives were intended to support the development of new (own) diagram types. They, however, used different kinds of configuration files to define the elements to have graphical counterparts and to be used to generate a starting point for manual diagram type development (programming). In both approaches, the configuration files refer to metamodel concepts so that code created based on them refers to code created to correspond to the metamodel concepts.

As a metamodel for graphical support generation, it would be possible to use both a new (MOF) metamodel and UML metamodel so that new concepts would be defined with Stereotypes. However, both GEF/GMF and Topcased based approaches are intended for building diagrams on (MOF) metamodel elements. In, for example, the diagram configuration files of Topcased, diagram elements refer to metaclasses in the (MOF) metamodels, not to Stereotypes that could be applied to run-time instances of UML metaclasses. Checks for Stereotype applications could be added to the automatically generated code manually, in order to support the Stereotype based approach. However, it could require error-prone switch-case (or e.g. if-else) structures to query applied stereotypes and other similar changes to several places in generated code that could be difficult to be kept up-to-date.

---

[5] http://wiki.eclipse.org/GEF

[6] http://wiki.eclipse.org/GMF

[7] http://eclipse.org/modeling/

3.3.3 Model Transformation Techniques

In addition to graphical development, selection between the extension mechanisms affects the use of model transformations. Standard QVT model transformations are naturally suited for the metamodeling (MOF) based approach. This can be understood based on relationships between models, metamodels and model transformations in Figure 1 in Section 2.1.4. MOF based metamodels are on layer M2 so that concepts in them can be accessed from transformation definitions on layer M1. Profile models with Stereotype definitions, however, would be on the same layer with the transformation definitions, and could not be accessed from transformation specifications.

The stereotype applications and tagged values of UML models can be queried from transformations with, for example, OCL [56]. However, the use of Stereotypes in transformations would require defining e.g. switch-case structures based on stereotype and property names. A transformation programmer would need to know the exact names of the stereotypes and their tagged values. Programming-time type checks would not be available in addition to, for example, auto correction functions. This is because the profile models would not be actually used until executing the transformation. With a static metamodel, for example, correction functions and consistency checks are possible. When compiling a transformation, the contents of it can be compared with the names and concepts of the metamodel.

3.4 UML AP Tool Implementation

In the tool development, a profound decision was the selection of an existing tool to be extended, which was made in order to re-use the support of an existing tool for plain UML and SysML. It was, though, assumed that the tool to be extended should be an open source tool, so that modifications to existing functionality would be possible, if needed. Among suitable tools, the choice was Topcased. At the time of beginning the tool development, it was one of few tools supporting both UML and SysML and development of new diagram types [P1]. At the time, an alternative would have been the Modeling Project of the platform that was based on GEF/GMF techniques. However, in addition to UML, Topcased provided extensive support for SysML and was ranked as the best available UML tool for Eclipse in a VTT study [62], too.

The following sub-sections will discuss the tool development from the point of view of implementing the modeling concepts (metamodel), graphical support for the new diagram types and extension interfaces for model transformations. Support for the use

of design patterns and re-use of platform specific blocks will be presented in sub-sections as well.

3.4.1 Metamodel Implementation

The basics of the tool implementation, related to metamodel and graphical support development, are discussed in the included publication [P1]. In addition to the selection of an existing tool, an important decision was the extension mechanism to be used to implement the new modeling concepts of UML AP. The selected basic mechanism was the metamodeling based approach, with MOF. As discussed earlier, the MOF based approach has few restrictions when changes to modeling concepts are additions (instead of removing elements, for example), which was the case with UML AP. UML AP with its diagram types also required new meta-associations between metaclasses, which would have caused challenges with the Stereotype based approach. The metamodeling based approach is also well supported related to developing new (own) diagram types. The SysML metamodel used by Topcased, for example, has been implemented with Eclipse Modeling Framework (EMF) by extending the UML2[8] implementation of UML metamodel, on the platform.

The majority of the new UML AP concepts have, thus, been defined with EMF, which is a MOF implementation on the platform and used by several modeling tools. However, in addition to MOF based extensions, some UML AP concepts were implemented as Stereotypes. In this way, the concepts (Stereotypes) can be used also in UML and SysML models and diagrams without changes to their program code. [P1]

The developed metamodel, which specifies the new UML AP concepts, is dependent on the UML metamodel of the platform (UML2) so that concepts of UML can be used and extended by UML AP concepts. In addition, the metamodel extends and is dependent on Topcased implementation of SysML metamodel. The MOF-based extension approach was facilitated by the availability of the EMF models related to the UML and SysML implementations so that they could be referenced from the developed EMF model (which was a metamodel from the point of view of the tool development).

The generated implementation for the (EMF) metamodel is dependent on the respective (Topcased and UML2) plug-ins that implement the UML and SysML metamodels. Since only new metaclasses were required, instead of modifications to existing ones, the

[8] http://wiki.eclipse.org/MDT-UML2

additions could be realized in a distinct plug-in [P1]. The dependencies between the plug-ins implementing UML, SysML and UML AP metamodels are illustrated in Figure 4 in Section 3.4.2. The figure also illustrates the dependencies between the corresponding graphical editors.

The initial profile implementation, which is described in [P1], has been later extended with concepts related to, for example, the modeling of safety aspects, control logic and design patterns. These extensions are described in more detail in Chapters 4 and 5 of the thesis. These extensions to the modeling concepts have been implemented so that new elements have been added to the metamodel. The procedure has been to edit the metamodel (the EMF model), to re-generate an EMF generator model (genmodel) and to re-generate the implementation code (see Figure 3). After re-generating code, small manual modifications have been required related to, for example, the initialization process of (Java) classes corresponding to the metamodel elements.

The extendibility aspect was not included in [P1]. However, according to experience gained during the research, it has been possible to further extend and change the profile implementation with a reasonable amount of work. When changes have been limited to the additions of new metaclasses, old code related to graphical modeling, for instance, has also been possible to be re-used without changes.

3.4.2 Graphical Support for UML AP Diagram Types

The graphical support of the tool was initially developed to implement the new diagram types of UML AP, namely Requirements Specification Diagram, Control Structure Diagram and Automation Sequence Diagram [P1]. All these diagram types are also needed in the AUKOTON development process. Requirements Specification Diagrams are used during the requirements phase and Control Structure as well as Automation Sequence Diagrams during the functional platform independent and platform specific design phases. In the included publications, the graphical support development approach is discussed in [P1]. After the AUKOTON project and publication [P1], additional graphical support has also been developed for Logic Diagrams as well as for presenting risks and hazards with the Fault Tree Analysis (FTA) notation [P6]. Support for visualizing design patterns has been developed to be used in conjunction to all diagram types [P7].

At the beginning of the tool development, UML AP did not strictly specify the concrete syntax of the new diagram types and graphical presentation of all the elements. Instead, the initial specification provided few example diagrams. The intended users of the tool,

however, were automation and control engineers that are accustomed to traditional diagram types of the domain. Accordingly, the diagram types were implemented to resemble traditional diagram types of the domain, with the intention to help the intended users to familiarize themselves with the tool and the profile [P1].

The extended open source UML/SysML tool, Topcased, supports the development of new diagram types with specific configuration files, which are in [P1] called generator models. They can be used for generating graphical editor plug-ins, plug-ins that implement diagram types [P1] as well as, for example, plug-ins that contribute to the properties view of the platform. The generated diagram type skeletons can be further tailored [P1], for example, to modify the symbols of the model elements in diagrams. Assuming that a new metamodel is used as a basis of a new diagram type, Topcased configuration files can be used according to the process described in [63] and illustrated in Figure 3.

The metamodel is in the process defined with an EMF (ecore) model that is used as a basis for creating a genmodel and generating the implementing code for the metamodel. The genmodel is also required for creating an editor configuration and diagram configurations, with which it is possible to define editor properties and diagram types. Based on the genmodel, editor and diagram configurations get the information about the related metaclasses and to which (Eclipse) plug-ins and (Java) Packages the implementing code (for the metamodel) is generated. However, after generating a diagram type, for example, the metamodel can be changed and the implementation re-generated, provided that the classes that the diagram requires are in the same plug-ins and Packages. Especially, although UML AP metamodel has been changed, the additions of metaclasses have not broken existing diagram type implementations.

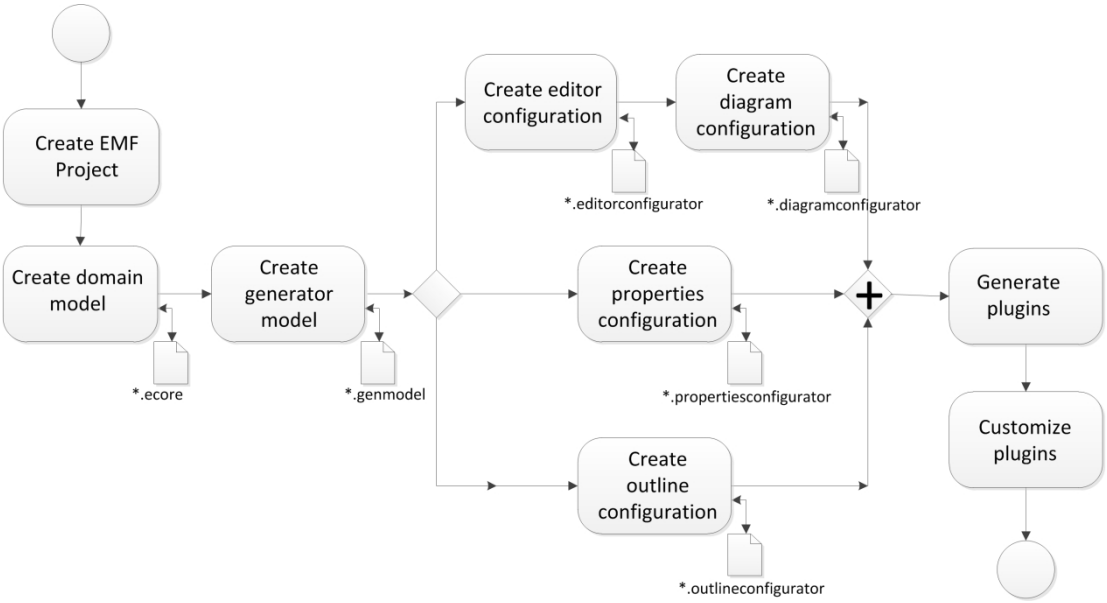

Figure 3 Graphical tooling development process with Topcased tool. (Modified from [63])

With the generation process, editor and diagrams become dependent on the metamodel implementations as illustrated in Figure 4. However, UML AP tool editor is also dependent on SysML and UML metamodels, in addition to UML AP metamodel. In a similar manner, the editor of the Topcased SysML implementation is dependent on both SysML and UML metamodels. [63]

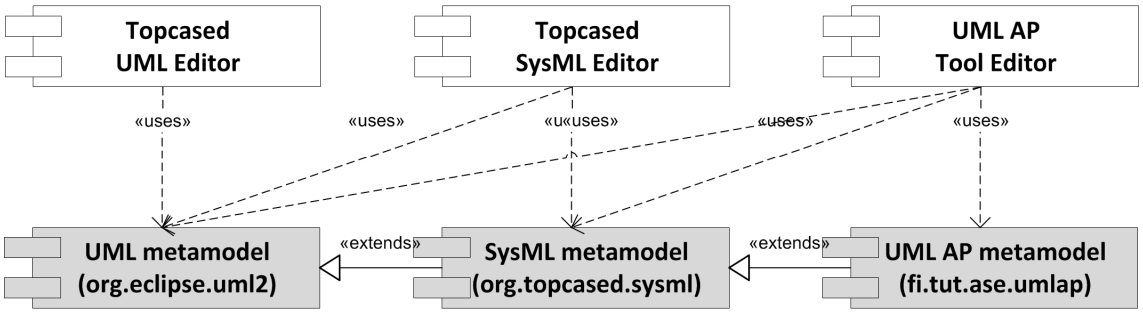

Figure 4 The dependencies between Topcased UML and SysML editors, UML AP tool editor as well as UML, SysML and UML AP metamodel implementations (Modified from [63])

3.4.3 Finding, Using and Controlling Model Transformations

As discussed, the AUKOTON development process requires three kinds of model transformations: import transformations, intra-model transformations and export transformations. These transformations differ from each other with respect to the purpose to which they are used. However, they also accept different parameters. Import transformations are targeted to a model Package, export transformations read the contents of a model Package and intra-model transformations are targeted between

Packages of a model [15]. In order to be able to support e.g. new source information formats, transformations should be added to the tool environment in a flexible manner [P1].

The mechanism to connect transformations to the tool environment is presented in detail in publications [P1] and [15]. The extension interface of the tool consists of three well defined extension points for the transformations. The extension point mechanism of the platform, in short, allows tools and plug-ins to search and consume one another's services without compile-time dependencies from the service consumers to the service producers [15]. With the mechanism, a plug-in can define an extension point to which other plug-ins provide their services. Defining such an extension point can include, for example, the specification of an interface that a plug-in implementing the extension must implement to provide the service. The platform, on the other hand, allows plug-ins to search and activate other plug-ins that implement extensions to such extension points.

In UML AP tool, the described mechanism has been used by defining an extension point for each type of transformation required by the AUKOTON development process. The tool, thus, defines a separate extension point for import, for intra-model and for export transformations. For each extension point, the tool defines a Java interface with appropriate operations that are used to control and to target the transformations to user-selected model packages. The interfaces are presented in Figure 5. In the figure, IRequirementImporter, ITransformer and IModelExporter interfaces are related to import, intra-model and export transformations, respectively [15].

Each interface defines two operations; one for getting a description of the transformation (to be shown for the user of the tool when selecting a transformation) and one for initiating the transformation [15]. The (UML) Packages to which references are relayed through the operations are Packages that the user of the tool has selected from the outline view of the tool when initiating the transformation [P1]. A relayed Package can also be an instance of the Model metaclass, which are used to contain whole model structures. In addition, the operations relay references to "traces" and/or "src" folders of the (Eclipse) workspace project that contains the model. Those references can be used for saving traceability information and for storing generated code, for example [15].

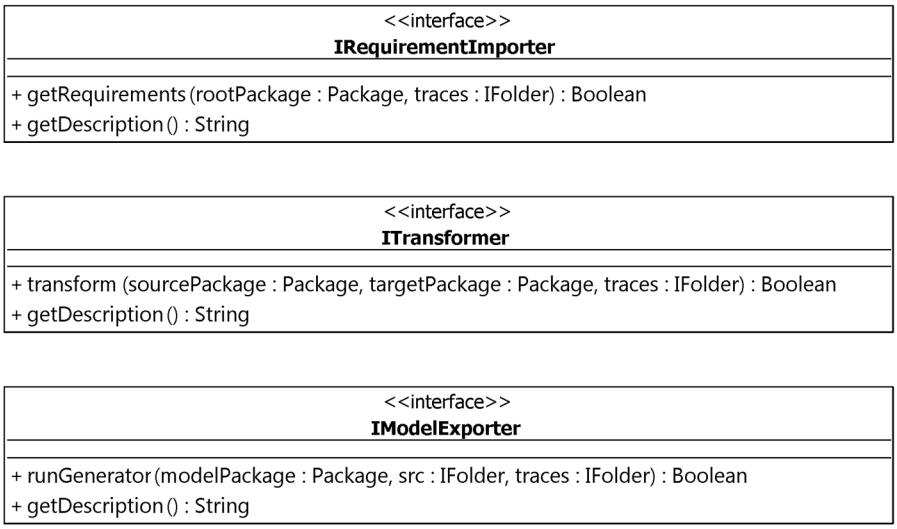

Figure 5 Java interfaces related to the extension point for import, export and intra-model transformations.

With the extension interface, controlling model transformations can be done with the graphical user interface of the tool while at the same time selecting the parts of models to be processed [P1]. The provided references to model Packages do not restrict access (from transformations) to the selected Packages only [P1]. Instead, in order to protect models from simultaneous modifications, the user interface of the tool is locked when a transformation is being performed. However, an approach to implement such a restriction has been presented in [15]. In addition, [15] discusses extending the use of the transformation-related extension points to other tools and presents a plug-in structure for SmartQVT model transformations. The plug-in structure has been utilized in code generation used in [P2] and in simulation model generation [P3], [P4]. The extension points themselves have been used in [P2] related to all transformation types, and in [P3], [P4], [P6], [P7] and [P8] related to export transformations.

Finding model transformations is automated by the tool. When a user of the tool initiates an activity to perform a transformation, the tool uses the platform services to find plug-ins that implement extensions to the transformation-related extension points. Found plug-ins are activated so that their descriptions can be loaded and provided for the user, who can then select between available transformations or choose not to perform a transformation.

3.4.4 Design Patterns in Modeling

Re-use of designs and solutions is an important means to enhance both the efficiency of development work and quality of developed applications. A means to document proven

solutions are design patterns, which capture knowledge for re-use purposes, for both expert developers and less experienced ones. In the included publications, the developed support for design patterns is presented in [P7]. The work presented in [P8], on the other hand, extends the pattern support for safety systems, with the objective to use patterns for producing safety documentation.

UML, with its existing modeling concepts, aims to support the use of design patterns in models [P7]. However, the support is restricted by nature and focused on the Classifier concepts of UML, leaving other important aspects outside the pattern support. In UML, patterns are defined with the Collaboration concept that extends both StructuredClassifier and BehavioredClassifier concepts. A pattern is a set of cooperating participants, which are Properties of the Collaboration. Pattern instances, on the other hand, are presented with CollaborationUses. CollaborationUses are owned by Classifiers to the contents of which the patterns are applied [P7].

Pattern literature of today, however, is not limited to the contents of Classifiers (Classes) only. For example, patterns can be on an architectural level and related to organizing an application to layers [64]. This could be in UML models described with, for example, Packages or Components, but not with the pattern modeling concepts [P7]. For example, a structure in Figure 6 could not be marked as a Layers instance. The Packages (Layer 1, Layer 2 and Layer 3) are not UML Properties or contained by a UML Classifier that would have to contain a CollaboratioUse element and all the role elements (layers) related to the design pattern instance. In a similar manner, it would not be possible to use Collaboration and CollaborationUse concepts to describe the contents of a UML class diagram. Class (definitions) would not be Properties or contained by a Classifier [P7]. Nevertheless, if design patterns are utilized in a software project, documenting their use in models could be of great value. On the other hand, if, for example, classes are deliberately designed so that they can be used according to a pattern, it should be possible to mark the intentional use of the pattern [P7].

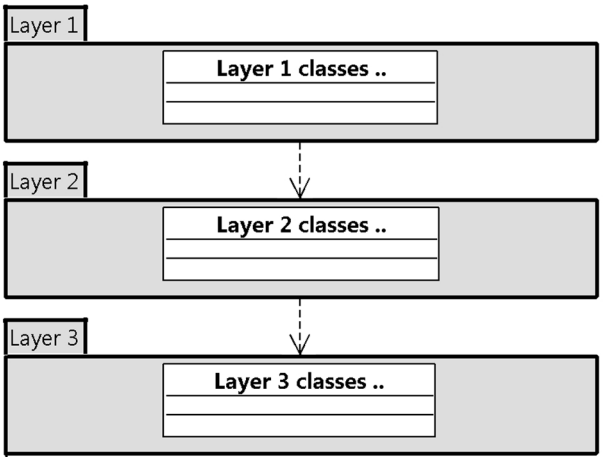

Figure 6 A presentation of Layers pattern with UML class diagram. (from [P7])

In addition to the standard approach, many tool vendors have developed pattern support in a more ad hoc manner. Magicdraw, for example, enables the use of informal UML templates that can be copied to models in order to instantiate patterns. However, as discussed in [P7], in this way the information on pattern instances is endangered to vanish and pattern occurrences can be difficult to notice for both developers and tools. On the other hand, UML concepts are suited to provide only information on solution parts of patterns leaving e.g. the problem parts unspecified [P7].

The developed pattern modeling approach is presented in detail in [P7]. The approach is aimed to be less restrictive than that of UML and to enable the specification of part of the information content that UML leaves intact. Specification of contexts and problems, which are essential information content of patterns [59], are enabled with text (attributes) in addition to the names and solutions of patterns [P7]. The metamodel of the pattern modeling concepts is presented in Figure 7 that has been divided into two parts. The concepts on the left-hand side are aimed for defining patterns and for organizing related patterns whereas the concepts on the right-hand side are aimed for marking pattern instances. It is also foreseen that patterns could be defined in specific library models in a similar manner than, for instance, stereotypes are defined in profiles by domain experts and then used in a number of application models.

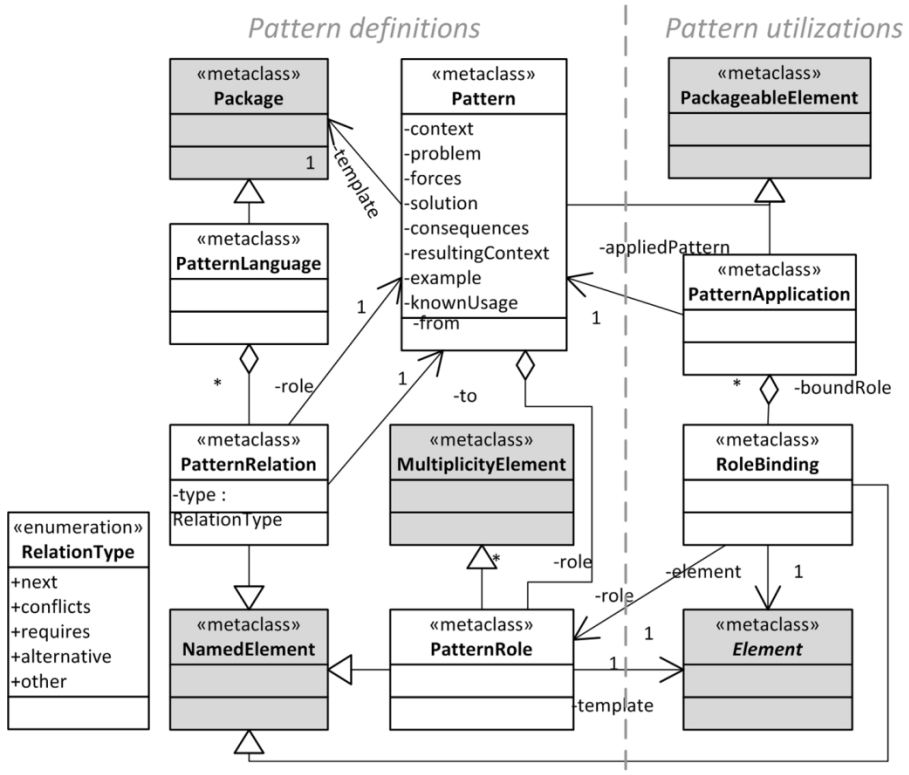

Figure 7 Metamodel of the pattern modeling concepts. (from [P7])

The Pattern and PatternApplication concepts are aimed for defining patterns and for marking pattern instances, respectively. Patterns consist of pattern roles and contain textual information, which has been structured based on the canonical form of patterns [65] with the addition of consequences of the pattern form in [58]. The Pattern concept is extended from the UML PackageableElement concept so that Patterns can be defined in Packages or PatternLanguages. Patterns and PatternRoles can also refer to template elements, which can be used for automating the application of patterns. PatternApplications, on the other hand, are used when applying patterns. As a difference with regard to the UML concepts, pattern instances need not be owned by Classifiers but Packages, which are used in models in any case. Model elements that play roles in a pattern instance can be any direct or indirect contents of the Package, instead of properties of Classifiers only. The role elements are tied to pattern specific roles (PatternRoles) with RoleBindings [P7].

In addition to the concepts, tool support has been developed to instantiate and to visualize patterns in models as well as to generate documentation from models in which the concepts are used [P7]. Defining a pattern with the concepts has to be done only once for each new pattern. However, because configured PatternApplications are required for each pattern instance, it is natural that the task should be automated, in

order not to require additional work from developers [P7]. In the tool, the task is included in a wizard with which it is possible to create pattern instances, by copying role elements from templates or by selecting existing elements for pattern specific roles. Markings are created as a by-product of applying patterns with the wizard.

Pattern markings support the traceability of solutions in models. With specific pattern concepts, it is also possible to query models on the use of patterns and to gather statistics on their use. For example, in [P7] documentation generation functionality is presented that collects information on the use of patterns and their (backward and forward) traceability to MS Excel (spreadsheet) documents. Tool support for benefitting from design patterns includes also a function to visualize pattern instances in models with the same (Collaboration) notation that UML uses. However, to keep the amount of details in diagrams sufficient, patterns are highlighted only when requested [P7].

An example diagram with a highlighted Observer [26] pattern is presented in Figure 8. Diagrams with visualized pattern instances can improve the documentation value of diagrams by making use of standard solutions explicit [P7]. With the visualizing support for patterns, it is also possible to use existing models as training material so that it is easy to find out how and in which situations the patterns have been used.

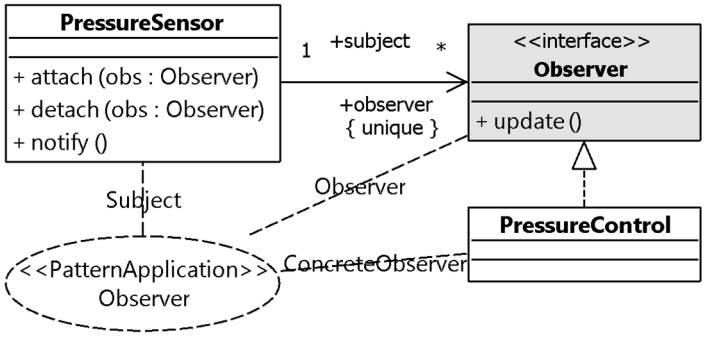

Figure 8 A visualization of an Observer pattern instance. (from [P7])

3.4.5 Platform Specific Implementation Blocks

In addition to design patterns, re-use in the domain can be related to implementation blocks that can be collected into libraries and used in different applications. Such blocks are often called type circuits. They are often platform specific, so that their utilization can be considered enhancing re-use in the functional, platform specific design phase of the AUKOTON process. Controllers that implement the well-known Proportional-Integral-Derivative (PID) control algorithm, for example, are by nature re-usable with only changes to their tuning parameters. In the included publications, an overview of the

platform specific modeling approach is provided in [P2]. In addition, the approach is presented in detail in [51] as a part of the whole AUKOTON process.

The purpose of the platform specific modeling phase (of AUKOTON) is to detail functional, platform independent design to a platform specific level, so that code generation can be performed based on the models [P2]. To achieve this, the development process relies on platform specific profiles that are developed to support control system platforms and their existing collections of type circuits. The platform specific profiles contain Stereotypes corresponding to the type circuits of the platform in question [P2]. Tagged values (properties) are added to the Stereotypes so that they correspond to the parameters of the type circuits, and can be set in the models. Lastly, the signal interfaces of the type circuits are defined in template AFs with ports so that signal interfaces of AFs in application models can be compared with those of the templates.

For example, in the assessment of the AUKOTON development process and tools [P2], a type circuit collection AUKOTON DCS was used. The collection included FBs in PLCopen IEC 61131-3 XML format and had been modeled as a platform specific profile - with stereotypes, tagged values and template AFs. Stereotypes and their tagged values related to the AUKOTON DCS collection are shown in Figure 9. Figure 10 visualizes the template AFs related to LC_3 (Limit Controller 3) and PIDC_2 (PID Controller 2) type circuits including the available ports in their signal interfaces. During the platform specific assessment phase [P2], the platform independent functional model was completed with AUKOTON DCS specific stereotypes and ports. The stereotypes were used to map the AFs to the existing type circuits (function blocks). The signal interfaces of the AFs were compared with those of the templates and completed to correspond to them, when necessary. This is in the tool environment an automated function and implemented by automatically copying missing ports and warning about additional ones that do not exist in the templates. After performing this, it was possible both to use type circuit specific parameters and their port interfaces [51].

Based on such platform dependent models, the process of generating executable applications is straight-forward. The process includes instantiating (existing) type circuits, linking the instances together based on connections in the models and specifying their parameters based on tagged values in the models. In general, this process is also far less error prone than, for example, constructing applications from programming language level concepts [51].

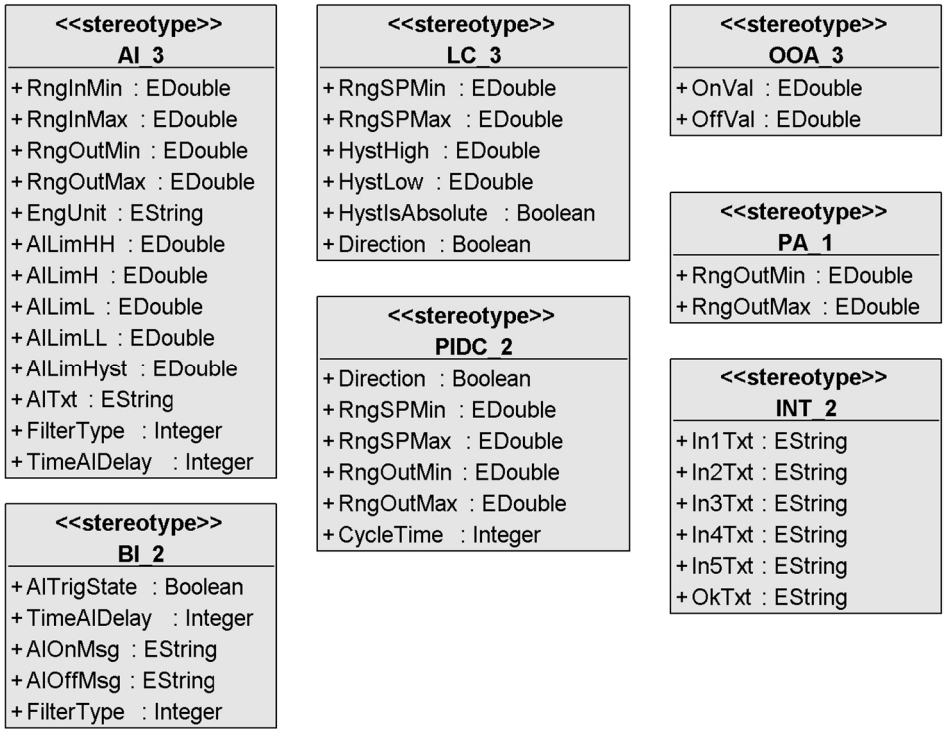

Figure 9 Stereotypes and their tagged values related to the FB collection used in [P2]. (Modified from [51])

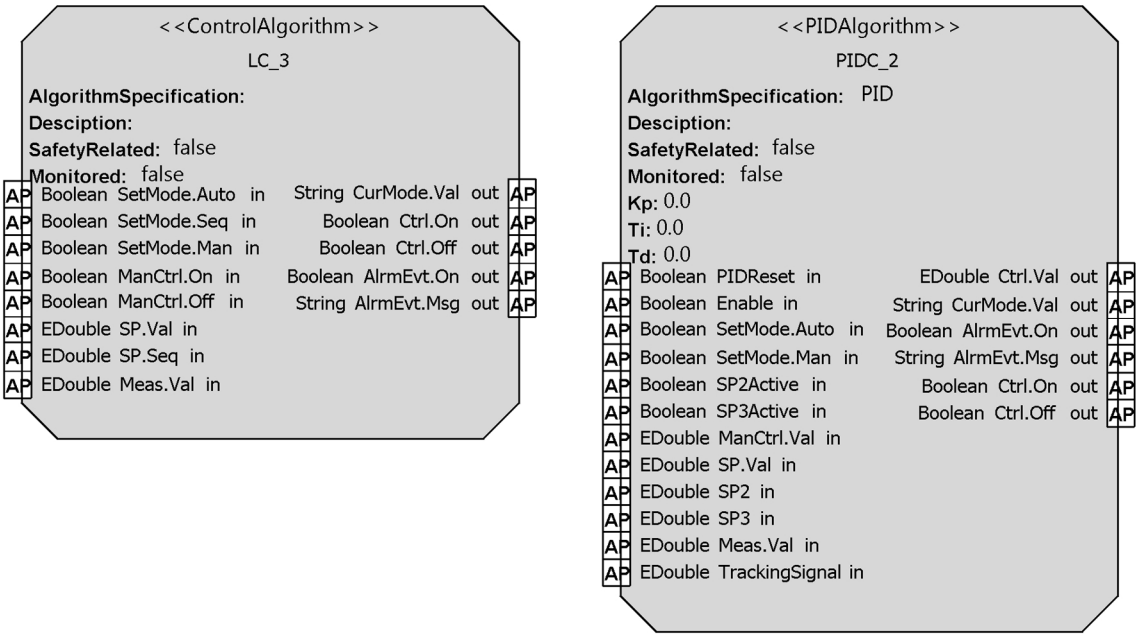

Figure 10 Template AFs related to LC_3 and PIDC_2 type circuits that were used in [P2].

For [P2], the approach was successfully applied to generating code in PLCopen XML format, with the AUKOTON DCS type circuit collections. However, in addition to producing IEC 61131-3 based PLC applications, an industrially applicable development

process should support proprietary DCS platforms and their collections of type circuits [P2]. Without resources to implement code generation for various target DCSs, the possibility to support DCS platforms has been assessed only based on interviews with the participants of the assessment event [P2]. The interviews suggested that the approach that is based on stereotypes and ports could be used also for platform specific design and code generation for certain proprietary DCS platforms. Identified areas of further development included the capability to specify interlocks in a detailed manner, replacing UML terminology with domain concepts and ability to visualize and edit designs at different levels of granularity. These further development areas have also been addressed in later work. For example, support for detailed logic diagrams has been added [P6] and used also for simulation purposes [P3, P4].

3.5 Discussion

The UML AP tool has been implemented by extending an open source UML/SysML modeling tool, Topcased. The majority of the modeling concepts were implemented with the MOF based extension approach. However, some UML AP concepts were implemented as UML Stereotypes so that they can be used in UML and SysML models and diagrams, which are in the tool environment re-used from Topcased. In this way, the profile implementation could be kept straight-forward and required meta-associations between metaclasses could be easily implemented. As a whole, the approach proved it possible to extend UML on both M1 and M2 metalevels with reasonable resources.

The approach enables the co-use of modeling languages and profiles so that UML, SysML and UML AP concepts are used in the same models. Since only new metaclasses were required to implement UML AP, instead of modifications to existing ones, the metamodel additions could be realized in a distinct plug-in that extends and is dependent on UML and SysML metamodel implementations. The implementation is extendable for future needs, in a similar manner than the UML and SysML metamodels were extended. It has also been possible to extend the profile implementation along the research without changes to, for example, existing diagram types.

An assessment of industrial applicability of the development process and tool has been presented in [P2]. According to the industrial feedback, the process and tools could be used for developing industrial DCS based control applications. The process and the tools enable automating part of the design activities that are performed currently manually but with the cost of introducing an additional work phase with requirements.

The use of metamodeling techniques to implement the concepts of the profile enables developing model transformations with standard transformation techniques, e.g. QVT. While model processing and transformation techniques, e.g. QVT and OCL, would enable querying models about Stereotype applications, their support for transformations between metamodel concepts includes additional consistency checks. In this way, the concepts of the metamodel can be used in the transformation definitions.

To facilitate the use of model transformations, which are required by the AUKOTON development process, the tool defines its own extension interface. The extension point mechanism of Eclipse allows tools to search and consume the services of other tools and plug-ins without compile-time dependencies to service providers. Especially, to support transformations, UML AP tool utilizes the mechanism with three extension points, one for each transformation type required by the AUKOTON process. With the mechanism, plug-ins implementing transformations can be added to the tool in a flexible manner, for example to support new source information formats. Searching and controlling model transformations can be done with the graphical user interface of the tool.

Each extension has been defined with a Java interface that an extending plug-in must implement. The interfaces, then, include operations that are required to get information about the transformations and to initiate and target transformations to appropriate model Packages. The extension interface has been utilized successfully in research related to [P2], [P3], [P4], [P6], [P7] and [P8].

The design pattern support of the tool has been developed to increase re-use on a platform independent level and to complement the pattern support of UML. The need for the new pattern modeling concepts originates from the UML pattern support that restricts patterns to describe the contents of Classifiers. The pattern literature of today, however, is not restricted to such a narrow scope and includes patterns on the architectural level, for example.

The new pattern modeling concepts enable the definition of patterns in specific library models and marking of pattern instances in application models. The concepts relieve the impractical restrictions of UML concepts. Patterns are not restricted to describe the contents of Classifiers but Packages, which are used in models in any case, and patterns can consist of instances of practically any metaclass. In addition to the concepts, tool support has been developed to instantiate and to visualize patterns in models as well as to generate documentation from models in which the concepts are used. Patterns and their instances both promote the re-use of known solutions and support traceability

between solutions and their use in system designs. This traceability information can be queried and collected into MS Excel spreadsheet documents.

To enable re-use of platform specific implementation blocks, a hybrid modeling approach was chosen. Implementation blocks and their platform specific parameters are modeled as Stereotypes and tagged values related to the Stereotypes. Automation Functions can be applied with (platform specific) Stereotypes to select library blocks to be used in the places of the Automation Functions in platform independent models. By applying a Stereotype, it becomes possible to set the block specific parameters with tagged values. The signal interfaces of the blocks, on the other hand, are modeled with template AFs. Automated functions have been developed to compare the interfaces of AFs with the interfaces in the templates and to complete them, when necessary, so that implementation block specific connections can be used in models.

With the approach, code generation activity can be implemented to instantiate, connect and parameterize blocks according to models. For [P2], for example, a function block collection that had been implemented with IEC 61131-3 was modeled as such a platform specific profile. The collection was then utilized for successfully generating a function block application in PLCopen IEC 61131-3 XML format.

4 Simulations in Model-Driven Development of Control Applications

The general simulation approaches that can be used in MDD to assess the models that are used in the process and produced applications include Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL) and Hardware-in-the-Loop (HiL) simulations [17]. In control application development, the differences between the approaches are in the control application (or control system) counterparts that are used to control the plant[9] (process) parts of the closed-loop simulations. Accordingly, the approaches also differ with respect to the nature of problems that they can reveal and with respect to the design phase in which it is possible to apply them.

For example, a MiL simulation using (only) a model of a control application to control a plant model is capable of validating the holistic control solution. SiL, PiL and HiL simulations use software generated from the models, generated software with target processors and generated software with the entire target control system hardware, respectively. In addition to the control solutions, these approaches are able to evaluate also other aspects in the designs. However, they also require more design phases (such as hardware design) to be completed and thus may not be performed as early during the development as MiL simulations.

Another classification of simulation approaches is related to the amount of simulation engines. A closed-loop simulation can be performed within a single simulation engine or as a co-operative simulation (co-simulation) by simulating the parts of the overall simulation model in different but connected environments. Co-simulation, however, naturally requires a mechanism to connect the environments and to replicate the simulation commands to them. These tasks are addressed in Functional Mock-up Interface (FMI) standard [66] and have been recently approached also with model-based techniques [67].

In MDD in the domain, the use of simulations has been integrated into several recent approaches in order to be able to validate designs early. In [68], Hegny et al. use an IEC 61499 runtime for simulating both the control application and plant parts of models. The behavior of a plant model is described within a composite function block using either a

[9] Also hybrid plant models that consist of real and simulated parts are possible.

timed state chart or an external behavior description, the latter alternative making the approach a co-simulation approach.

Yang and Vyatkin [69], similarly, rely on IEC 61499 but create plant simulation models with the use of model transformations, by transforming Matlab/Simulink plant models to IEC 61499. They thus enable closed-loop simulations within a single simulation (or IEC 61499 runtime) environment. In [35], Vyatkin et al. propose a model-integrated design framework. In the framework new systems should be based on intelligent mechatronic components that would include software components as well as models that enable system simulation and formal verification. The simulation and formal verification of control applications are also envisioned in [70], by Vyatkin et al.

In [71], the aim is to mix real control hardware and software with simulated ones while simulating the plant model in another simulation environment. The benefit of the co-simulation approach is the ability to test early and concurrently with the engineering work. In the FLEXICON approach, co-simulations are enabled with Data Distribution Service (DDS) middleware between the tools [72], and Common Object Request Broker Architecture (CORBA) in the previous version of the approach [39]. In DECOS [73], simulations are enabled by the modeling techniques that are used for modeling application behavior, e.g. Matlab/Simulink and SCADE.

Model-based approaches to the simulation-assisted evaluation of control applications have also been enabled in commercial products. For example, Beckhoff[10] and Bachmann[11] have products for generating PLC code based on Matlab/Simulink models. In application domains other than industrial control, e.g. automotive control systems, it has also been common to use simulations and simulation-aided testing within model-based development. A general framework for and two examples of use of MiL simulation and testing have been presented in [74]. A testing environment for embedded systems with SiL simulation has been presented in [75]. HiL simulation and testing have been utilized, for example, in [76] and [77].

This Chapter discusses the use of design-time simulations in model-driven control application development and is organized as follows. Requirements for the use of simulations are presented in Section 3.2. Possible simulation approaches and implementation techniques are discussed in Section 3.3. Section 3.4, then, presents the

[10] http://www.beckhoff.fi/english.asp?twincat/te1400.htm?id=1889849218919049

[11] http://www.mathworks.se/products/connections/product_detail/product_35950.html

developed simulation approach including observations from applying it in several publications.

4.1 Requirements for Simulations in Control Application Development

4.1.1 Benefits of Simulations in Control Application Development

One of the key promises of MDD is the ability to automate simple, repetitive design tasks with model transformations. However, demanding design decisions over alternative solutions to achieve (sometimes informally specified) objectives and product characteristics are still made by professional developers. Fortunately, such decisions do not need to be made based on developer experience only. Already with more conventional control application development approaches, the use of simulations has played a significant role in facilitating the decision making [P3].

Simulation solutions are also commonly provided by commercial DCS platform vendors [16]. For PLC based control systems, on the other hand, soft PLC solutions enable the execution of control programs on desktop computers. Benefits of simulations have also been reported by several researchers. According to a survey of Carrasco and Dormido [16], the benefits of using control systems in simulators before installation include improvements to 1) design, development and validation of the control programs and strategies, 2) design, development and validation of the HMI (Human-Machine Interface) and 3) adjustments of control loops and programs.

According to Dougall [78], the use of simulations enables better operator training, ability to test control programs in smaller modules and the ability to the thorough testing of emergency and dangerous situations. In addition, the use of simulations can result in shorter start-up times of plants and processes, reduced site time as well as less waste of material and end products during the start-ups. Karhela [79] mentions the use of simulations for, for example, control system testing, operator training, plant operation optimization, process reliability and safety studies, improving plants and processes, verifying control schemes as well as for start-up and shut-down analyses.

As argued in [P3], many of the mentioned benefits of simulations are related to engineering tasks that MDD alone may not affect. MDD can enable the use of intuitive models and diagrams as well as model transformations and model checks to automate repetitive tasks. However, the use of MDD techniques does not reduce the need to

validate and test designs and products. In addition, many of the mentioned benefits of simulations (above) are related to tasks that would be beneficial to perform at design time, if it were possible to simulate designs. In general, corrections to design flaws are often most beneficial to be made as early as possible so that they do not affect adversely on later designs and decisions. As a consequence, it is possible that many of the mentioned general benefits of simulations could be obtained also by using simulations to complement a MDD approach, if it was possible to simulate design models.

However, by integrating simulations into a MDD process, it could be technically possible to simulate earlier. As argued in [P5], the restrictions of early (MiL) simulations within a MDD process are related to missing hardware (including, among others, the user interface), which complicates, for example, operator training. However, control application (MiL) simulations can be carried out already before selecting a target platform and performing hardware design. MIL simulations can also be used in companies that perform outsourced development tasks and may not have access to the control system hardware, even if control hardware design was performed already.

Following is a brief discussion on the properties of simulations that are useful in MDD of control applications to achieve the mentioned benefits.

4.1.2 Required Properties for Simulations

The use of simulations should be enabled in MDD of control systems and applications. It should be possible to simulate designs in a timely manner and in a closed loop with simulation models of the processes to be controlled [P3]. It should be possible to acquire prompt feedback about solutions and to compare alternative approaches, structures, tunings and, for example, interlocks. Design flaws should be corrected as early as possible so that they would not affect adversely subsequent design phases. Naturally, the simulation approach should provide support for all the common aspects of basic control systems including binary and feedback control, control sequences as well as interlocks.

In MDD, simulation should be an effortless activity [P3]. As the purpose of MDD is to improve the efficiency of development work by automating repetitive tasks, it cannot be assumed that developers would create simulation models manually. Instead, simulations should be possible to be used as a continuous quality assurance method, without slowing down development. Because in MDD it is possible to automatically generate executable code, it should be possible to generate simulation models as well. The simulation models should, thus, be created automatically, as a side product of the

development process and based on design models that are created in the development process. On the other hand, performing the simulations should not require special simulation expertise - skills that all control application developers may not have.

Simulations should be possible to be performed early in the development process. In this way, simulations could facilitate understanding the net effects of requirements that originate from the various design phases preceding the application development. Developing applications on requirements that originate from various domains of engineering is a special characteristic of the domain [P3]. It is vital to notice the effects of requirements early so that possible inconsistencies do not cause expensive re-work. In a similar manner, simulations could make the effects of changes to high-level models visible for developers [P3].

In addition to early simulation, the simulation approach should take into account the special characteristics of the domain related to re-using implementation blocks. Since libraries of existing implementation blocks are commonly used in applications, similar libraries should be available for simulations. In this way, platform and vendor specific functions and blocks could be validated as part of the design. The correspondence between the reusable implementation blocks and their simulation counterparts could be verified as well, in order to increase confidence on the results of simulations [P3].

In case of the AUKOTON development process, the above would mean enabling simulations during both functional platform independent and functional platform specific development phases (see Section 3.1). Functional models, without platform specific blocks, can already contain a significant amount of functionality to be validated. With platform specific details, simulations could be used to evaluate the platform specific blocks, tunings and predicted performance of control solutions. The development process with the required simulation extensions is illustrated in Figure 11.

However, it is also obvious that all functions and blocks of control applications cannot be re-used from libraries. For example, interlocks and sequential control activities are often developed specifically for each application and thus cannot be (always) re-used. Therefore, the simulation approach should be capable of creating and using new simulation blocks in addition to using the librarized ones [P3]. With current UML AP, such interlocks and sequences are modeled with Logic and Automation Sequence Diagrams, respectively.

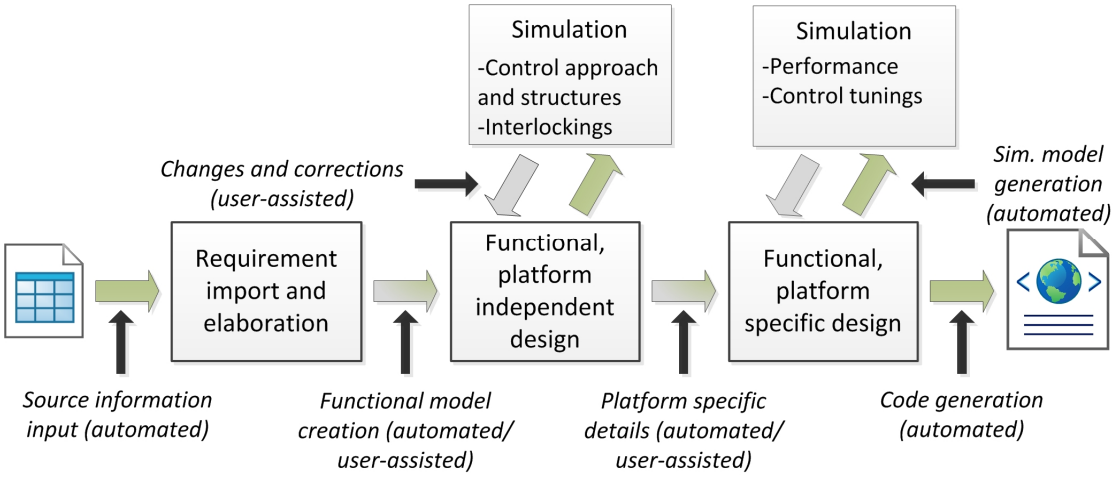

Figure 11 The AUKOTON development process with the simulation extensions. (Modified from [P4])

4.2 Considerations on Implementation Techniques

4.2.1 XiL Simulation Approaches

As presented, the general approaches for simulations in MDD include MiL, PiL, SiL and HiL, which differ from each other with respect to the control system counterparts that are used to control the plant simulations. Simulations are also supported by many control system platform vendors that enable connecting the control systems to simulators [16] in order to support PiL and HiL simulations. In these simulations, connections to simulations can be implemented programmatically or via I/O units. It is also common for industrial DCS vendors to support the computer execution of the control programs to enable SiL simulations. In addition, for PLC based control system platforms, there are soft PLC solutions that enable executing the control programs of PLC systems on desktop computers and thus SiL simulations [P5].

It can be, thus, said that SiL, PiL and HiL simulations are already supported by control system vendors. As a consequence, if simulations are to be used in a MDD process to develop control applications that are to be used in PLC or DCS platforms, developing support for other types of simulations than MiL might not be able to provide significant benefits. This is because the other types of simulations are (often) already enabled by the control system platforms and usable after generating code [P5]. By enabling the use of early MiL simulations, however, it could be possible to validate solutions (at least partially) before generating code or even selecting a control system platform and

hardware parts of it. As a consequence, in MDD in the domain the focus should be on MiL simulations [P5].

4.2.2 Number of Simulation Engines

As presented, a closed-loop simulation can be performed within a single simulation engine or as a co-simulation by simulating the parts of the overall simulation model in different environments. A MiL simulation can use either of the approaches. However, simulation types other than MiL are usually co-simulations of some kind so that the plant parts of the simulations are simulated in environments other than those acting as control systems. For example, coupling and synchronization mechanisms are thus required to connect the simulations (environments). In the domain, there have also been approaches (e.g. [68] and [69]) in which plant models have been provided as such or transformed to IEC 61499 models and coupled to control application parts. If IEC 61499 is considered as a modeling standard, these approaches can be regarded as MiL simulations using a single simulation engine.

An objective for the simulation approach was that it should not require control application developers to have special expertise related to simulations. Co-simulation, however, necessarily requires a mechanism for coupling the simulation environments and replicating commands of them [P5]. Use of a single simulation engine, without the need to couple simulators, can be seen as a less complex approach. However, also in case of a single simulation engine the plant and control system parts need to be connected, if this task is not performed automatically e.g. by a model transformation. Because of this possible need for couplings - and inspired by the recent advances related to co-simulations - the co-simulation approach will be regarded as an alternative in the following sub-section that compares the alternative approaches to implement MiL simulations. Such recent co-simulation-related advances include the FMI (Functional Mock-up Interface) standard [66] and publication [67] in which model-based techniques are used to facilitate the coupling of simulations.

4.2.3 On Creating Closed-Loop MiL Simulations

As presented in [P5], in addition to using co-simulation there are several approaches to develop MIL simulations to be performed in a single simulation engine. This often requires the availability of both control application and plant models in the same simulation language. Such simulations can be achieved by

- developing both the plant and control application models with the same simulation language,

- transforming both the plant and control application models to the same simulation language,

- transforming the plant model to the (simulation) language used to develop the control application model or,

- transforming the control application model to the (simulation) language used to develop the plant model.

Related to the first alternative, the author is not aware of a language that would be a feasible alternative for both plant and control application modeling and would integrate well with UML-based MDD techniques. For example, the language would have to be processable with (preferably standard) model transformation techniques but also enable information transfer and requirement modeling during the early phases of software development. On the other hand, the second alternative would require developing and keeping up-to-date two possibly complex model transformations. Because of the amount of complex transformations, it could also be prone to errors. As a consequence, only the third and fourth alternatives appear feasible. Related literature in the domain also includes examples on the use of the third alternative, see [69] and [68]. The co-simulation approach (with hardware included) has been used, for example, in [71]. Included publications [P3], [P4] and [P5] present and evaluate the work of the author that uses the fourth alternative.

The transformation based approaches (of the list above) and the co-simulation approach are compared at a conceptual level in [P5]. The results of the comparison, with respect to connectability and transformability requirements that the approaches place on languages and tools as well as work related to using simulations, include the following:

- Simulation engines need to be connectable only in the co-simulation approach, in which parts of the overall simulation are simulated in different environments.

- Compared with the transformation based approaches, the co-simulation approach may require additional work related to coupling simulations and managing simulation cases for several simulation engines. Such simulation

cases can evaluate the closed-loop system in different operation points, for example.

- Both control application and plant modeling languages must be transformable in the transformation based simulations since they are used either as source or target models of the transformations. With the co-simulation approach, the languages do not have to be transformable, provided that they are simulatable so that the models do not have to be transformed to a simulatable form first[12].

The results are also summarized in Table 2. In the table, PML and CAML refer to Plant Modeling Language and Control Application Modeling Language, respectively.

Table 2 A summary of comparison between co-simulation and transformation based MiL-simulations within a single simulation engine. (Modified from [P5])

Characteristic of the simulation approach	**Co-simulation**	**Transformation based MiL with single simulation engine**
Requires simulation tool connectability	X	-
Requires additional work with simulation cases	X	-
Requires additional simulation management	X	-
Requires the transformability of PML	-	X
Requires the transformability of CAML	-	X

With respect to the criteria above, the restrictions and requirements of the transformation based approaches (to MiL simulation using a single simulation engine) are the same. The approaches, though, place different simulatability requirements to the modeling languages. The simulatability requirements can also be avoided by using two transformations - to transform both plant and control application models to a simulatable form. The results of comparing these approaches, with respect to the number of required transformations as well as simulatability of languages, include the following:

[12] However, transformability of control application modeling language can be required by the MDD process used to develop the control application model.

- The approaches to transform plant models and to transform control application models require necessarily one transformation. If both models are transformed, for example because they are not simulatable, the number of required transformations is two.

- In the approach to transform plant models (to control application models) the control application modeling language must be simulatable but the plant modeling language does not. In the approach to transform control application models (to plant models), the plant modeling language must be simulatable but the control application modeling language does not. Transforming both the models (to a simulatable form) does not require the languages to be simulatable.

The results are summarized in Table 3. In the table, TPM, TCAM and TPM&CAM refer to Transforming Plant Model, Transforming Control Application Model and Transforming both Plant Model and Control Application Model, respectively.

Table 3 A summary of comparison between the transformation based MiL simulations within a single simulation engine. (Modified from [P5])

Characteristic of the simulation approach	**TPM**	**TCAM**	**TPM&CAM**
Number of required model transformations	1	1	2
Requires the simulatability of CAML	X	-	-
Requires the simulatability of PML	-	X	-

Based on the comparisons, it is difficult to draw conclusions on which of the approaches would be the most recommendable. In practice, also current plant engineering processes and possible available process simulation models would be relevant factors. The co-simulation approach requires more simulation management and coupling work than the transformation based approaches. It does not necessarily require model transformations. A transformation, however, is required if either the control application or plant model is not simulatable. The approach to transform plant models (to control application models) requires either simulatability of the control application model or an additional (second) model transformation. For example, this second transformation would be required in case of UML models that are usually not simulatable.

4.3 Model-in-the-Loop Simulating UML AP Models

The developed approach to simulate UML AP models, which is described in detail in publications [P3] and [P4], uses the general Model-in-the-Loop (MiL) simulation approach. A single simulation engine is used to simulate both control and plant parts of the closed-loop model. A model transformation is used to create a ModelicaML simulation model of the control application, which is integrated (by the transformation) to an existing plant simulation model. The result is a closed-loop simulation model of the controlled plant. With respect to the classification of approaches to create simulations, the approach thus falls to the category of transforming control application models.

The following sub-sections will briefly present the simulation language used in the approach, ModelicaML, and the approach to create and integrate control application simulations to plant simulations.

4.3.1 ModelicaML as a Target Simulation Language

Modelica is an object-oriented, equation based simulation language. The basic concepts of the language are (simulation) classes. Simulation classes contain properties, equations (that determine the values of the properties) as well as connectors with which simulation class instances can be connected together. Similarly to object-oriented programming languages, Modelica classes can inherit properties and, for example, equations of super classes. Simulatable Modelica models consist of simulation class instances that are connected together using their connectors. Classes can also consist hierarchically of other classes.

In many tools, Modelica models can be composed by instantiating and connecting simulation classes graphically whereas plain simulation classes can be defined with the textual syntax of the language [P3]. In addition to object orientation, an important feature of Modelica is acausality. Since models are mathematically described by equations, instead of, for example, statements that are applied in an order, the order in which the equations become defined is usually not relevant [P3]. The use of equations also improves the re-use potential of simulation classes since equations do not specify a data flow direction [80]. However, in addition to equations the language includes an algorithm concept, for performing calculations in which statements are applied in order [P4].

ModelicaML, on the other hand, is a UML profile for creating, reading, understanding and maintaining Modelica models with UML tools. [20] The profile consists of stereotypes and tagged values that correspond to the concepts and keywords of the textual Modelica language. With the use of the profile, Modelica simulations can be defined in UML models using suitable diagram types. For example, composite structure diagrams can be used for representing how a simulation class consists of instances of interconnected (other) simulation classes. The behavior of simulation classes can be defined e.g. with textual equations or UML state machines, the execution semantics of which have been addressed in [81].

Choosing Modelica as the target simulation language in the approach is a result of several factors. Modelica is genuinely object-oriented similarly to the future programming languages of control applications, e.g. the latest version of IEC 61131 [30]. Applications and simulation models can thus have a similar structure. Modelica promotes the use of libraries, so that platform and vendor specific blocks can be re-used. Existing model libraries that are part of the language also facilitate the development of plant (process) models. Modelica is defined as an open specification. In addition, the ModelicaML implementation of OpenModelica [21] has been implemented with the same modeling and metamodeling techniques and tools as the UML AP implementation [P1]. Both are based on EMF and UML2 plug-ins on Eclipse platform.

Benefits of the similar background of the implementations include the ability to use standard QVT [13] transformations for defining the transformation from UML AP to ModelicaML [P3]. The translation of ModelicaML models to textual (simulatable) Modelica models, on the other hand, has been made publicly available by OpenModelica [21].

4.3.2 General Simulation Approach

The simulation model generation approach assumes that plant models are provided as ModelicaML models. In such a model, the main simulation class can be specified, for example, with a composite structure diagram (UML) that describes how the system consists of its interconnected parts. The parts, instances of lower level simulation classes, can be defined in the same model or e.g. in library models. Both the class definitions and instances need to reference the ModelicaML profile, which defines the ModelicaML stereotypes that map UML concepts to Modelica concepts. The purpose of the transformation, then, is to add control application specific simulation classes and their instances to the plant model, parameterize the instances and connect the instances

to the plant simulation with equations. An example model structure before and after applying the transformation is presented in Figure 12.

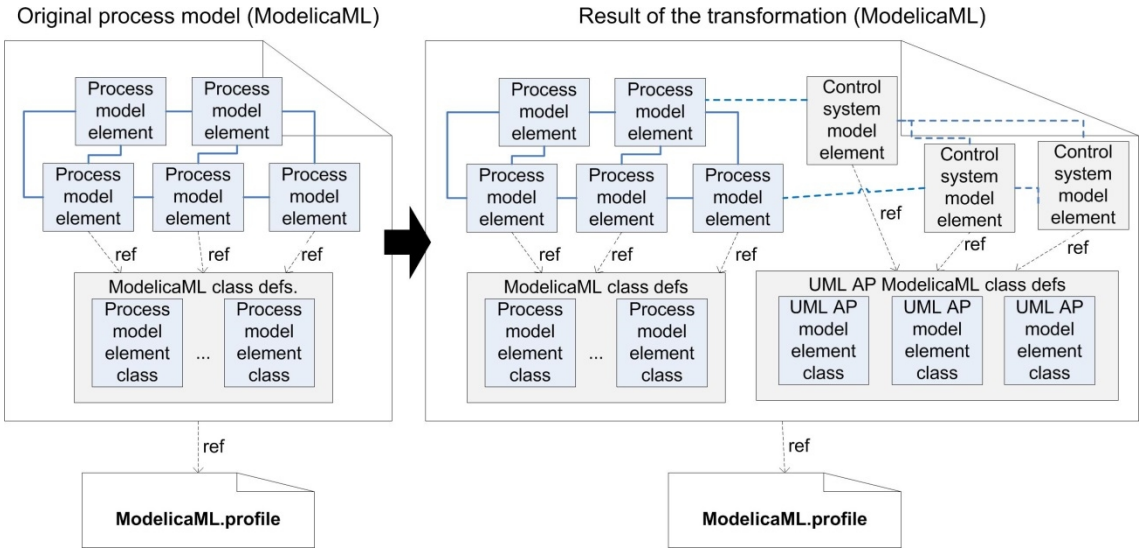

Figure 12 The transformation adds the control application specific parts to an existing plant model. (Modified from [P3])

In the process, simulation class counterparts of platform independent Automation Functions (AFs) are copied from a library contained by the tool [P3]. To support platform specific AFs, the transformation is capable of using external libraries that contain simulation class counterparts to such platform specific AFs [P3]. Lastly, to support application specific functionalities, the transformation is capable of creating simulation class definitions for AFs the functionality of which is described with Logic Diagrams [P3] or with Automation Sequence Diagrams [P4].

In detail the process that is illustrated in the figure above is presented in [P3]. Simplified, the process is performed as follows.

- The user of the tool initiates the transformation, selects the model Package to be exported to the simulation and selects the plant simulation model.
- ModelicaML class definitions corresponding to platform independent AF concepts are copied to the plant model from a library model.
- Platform specific AFs that are used in the control application model are identified based on the platform specific stereotypes that they apply. Simulation class definitions corresponding to them are copied to the plant model from the profiles that define the (platform specific) stereotypes.

- Application specific AFs, which are defined either with Logic Diagrams or Automation Sequence diagrams, are identified. Simulation class definitions corresponding to them are created. These processes are presented in detail in [P3] related to Logic Diagrams and in [P4] related to Automation Sequence diagrams. The processes are also briefly described in Sections 4.3.2.1 and 4.3.2.2, respectively.

- Instances of simulation classes are instantiated to the plant model according to the AFs in the control application models. In the first phase, they become properties of, for example, the main simulation class in the plant model.

- The types of the newly created properties are set so that they become instances of the simulation class definitions.

- The newly created instances of the simulation classes are connected together according to the control application model. Parameters, if any, are set based on the tagged values of platform specific stereotypes.

- Simulation class instances that interface with actuators and sensors of the plant model are connected to them based on the channel ID attributes of the AFs.

- Sufficient ModelicaML stereotypes are applied to the created model elements as required. For example, instances of simulation classes apply <<Component>> stereotype.

The process is thus quite straight-forward but sufficient for its purpose [P5]. The simulation approach also partially re-uses work presented in Chapter 3 of the thesis. Platform specific simulation class libraries corresponding to platform specific AFs can be added to the tool environment, similarly to the libraries of platform specific AFs, see 3.4.5. In this case, the library models should include the platform specific stereotypes, template AFs (for completing interfaces) as well as the simulation class counterparts of the blocks. Also the functionality to complete the interfaces of platform specific AFs can be used (as presented in 3.4.5) in order to use platform specific ports in models and simulations.

4.3.2.1. Processing of Logic Diagrams

In UML AP, Logic Diagrams can be used to describe the inner logic of AFs. A metamodel presenting the modeling concepts that can be used in the diagrams is

presented in Figure 13. The existing (related) UML metamodel concepts are in the figure presented with a gray color. As presented in [P3], all concrete AFs (in the metamodel) extend the abstract AF concept and are thus also kinds of StructuredClassifiers. Thus, they are able to contain Properties and Ports as well as Connectors with which Properties can be connected together. Since the Operations and the Constant concept of the metamodel are extended from the Property concept, AFs are able to contain instances of them.

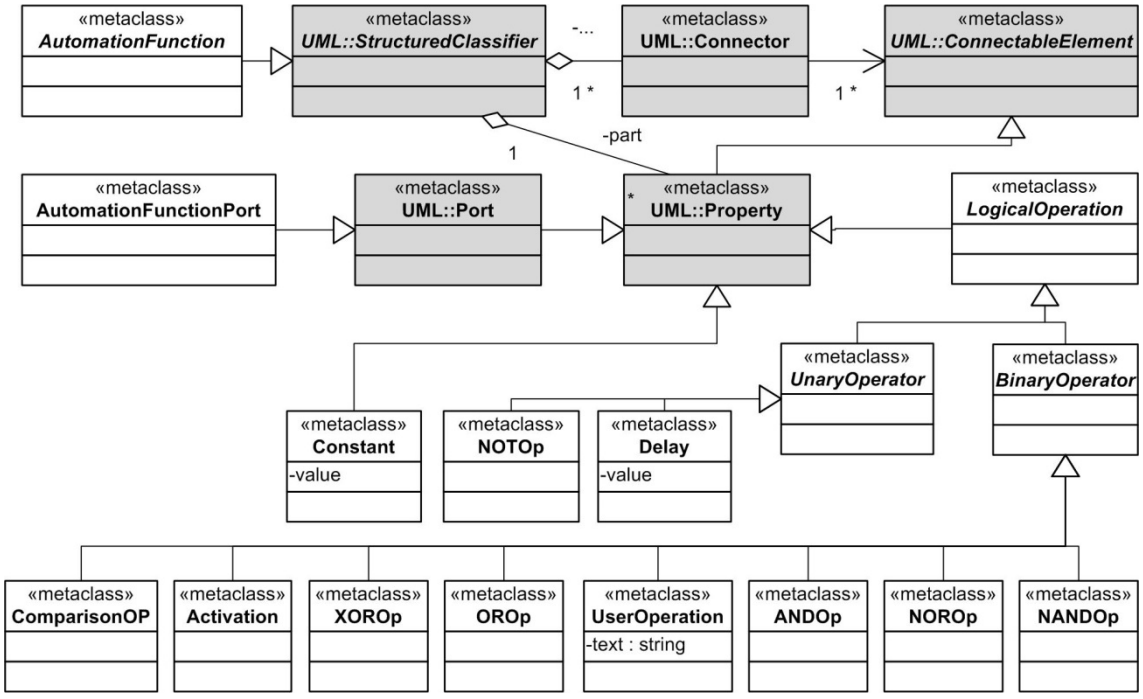

Figure 13 The metamodel of the Logic Diagram concepts including related UML metamodel concepts. (Modified from [P3])

The process of creating Modelica simulation classes based on Logic diagrams is by nature simple and described in detail in [P3]. Simplified, for each operation in a diagram the transformation creates a variable (Property) with a suitable (e.g. Boolean) data type. Equations to determine the values of the variables are then defined based on the metaclasses of the operations and Connectors coming into the operations. The Connectors can always be followed to other operations, which are in the Modelica models represented by other variables, or Ports. Simplified, the transformation is performed as follows.

- A new simulation class definition is created for each AF with a Logic Diagram definition.

- A Port with the same name and a corresponding type is created (to the class definition) for each Port contained by the AF.

- A variable (Property) with the same name and a corresponding type is created (to the class definition) for each LogicalOperation contained by the AF.

- An OpaqueBehavior element is created to the class definition to contain the equations (text) to be created to determine the values of the variables.

- The equations to determine the values of the variables are created based on the metaclasses of the LogicalOperations (e.g. OR) and Connectors coming into the Operations.

- The newly created elements are set to apply sufficient ModelicaML stereotypes, e.g. <<Model>>, <<ConnectionPort>> or <<Variable>>.

In the approach, the values of the variables are thus determined with equations. Since equations apply all the time, the order in which the equations become defined is usually not relevant. For example, the NOT operations in the Logic Diagram example of Figure 14 have been transformed to (Boolean valued) Properties the values of which equal to logical not operations of the values of the Ports (that are connected to the NOT operations). However, if Boolean valued loops (inside diagrams) are identified by the transformation, they are handled by creating algorithmic statements instead of equations. The order in which the statements should be applied is asked from the user of the tool. This is an interactive feature of the model transformation [P3].

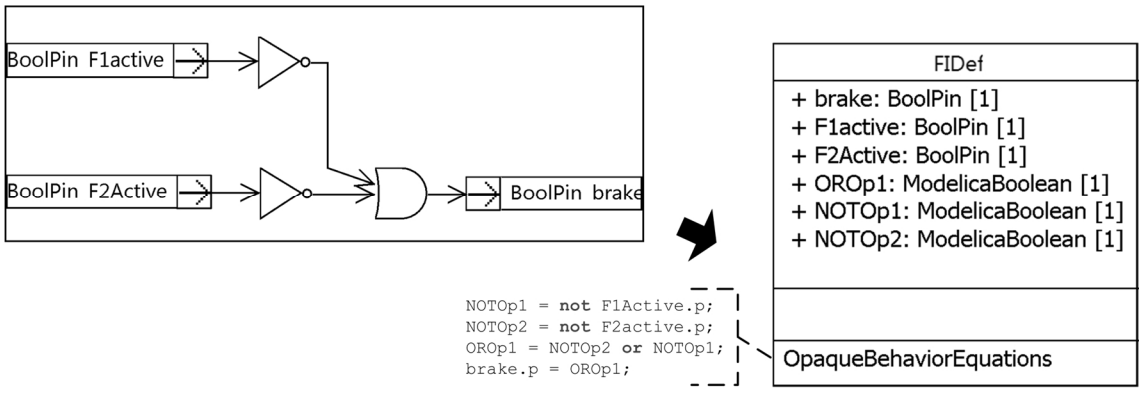

Figure 14 An example of transforming Logic Diagram to ModelicaML. (From [P5])

4.3.2.2. Processing of Automation Sequence Diagrams

In UML AP, Automation Sequence Diagrams (ASDs) can be used to describe the sequential behavior of AFs [P4]. A metamodel that presents the essential concepts that are used in ASDs is presented in Figure 15. The existing (related) UML metamodel concepts are in the figure presented with a gray color.

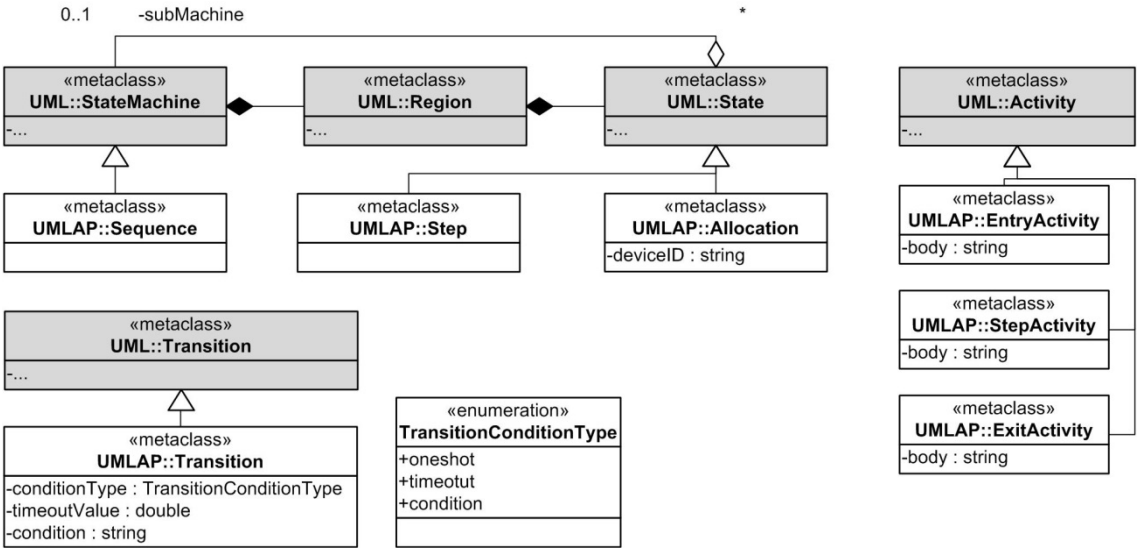

Figure 15 The simplified metamodel of the Automation Sequence Diagram concepts including related UML metamodel concepts. (Modified from [P4])

Sequences, which are the root elements of ASDs, consist of Steps that are the basic procedural elements in the approach. Similarly to states (of UML state machines) Steps contain EntryActivities, StepActivities and ExitActivities that are executed when arriving to the Step, during the Step and when exiting the Step, respectively. Steps can also reference other (sub) Sequences, which can be defined in other ASDs. Sequences can preserve process items and devices for their use with Allocations that are released at the end of the Sequences. The execution order of Steps in a Sequence is determined by Transitions as well as pseudo steps (which are not shown in the metamodel). Pseudo steps include initial and final steps as well as fork and join steps that can be used in a similar manner than the corresponding pseudo states of UML state machines. Transitions may also contain different kinds of conditions to control when they are fired.

Sequences, thus, represent the sequential behavior of AFs, which are represented by Modelica classes in simulations. To simulate the behavior of a Sequence, the transformation creates variables and algorithmic code to be owned by the Modelica class that corresponds to the AF that owns the sequence. The systematically named

variables are used to keep track of the execution of the Sequence. The algorithmic code, which utilizes element type specific code templates, on the other hand, changes the values of the variables according to the Sequence and performs the Activity code related to EntryActivities, StepActivities and ExitActivities. As described in detail in [P4], the variables are created according to the mappings in Table 4.

Table 4 Mappings between UML AP and UML (ModelicaML) model elements (modified from [P4]).

Source model (UML AP)	**Target model (UML with ModelicaML)**		
Element	**Element**	**Name**	**Type**
Sequence	Property	Seq. name	Boolean
	OpaqueBehavior	Seq. name + "Algorithm"	-
Step	Property	Seq. name + Step name	Boolean
	Property	Seq. name + Step name + "Phase"	Integer
Transition[13]	Property	Seq. name + Step name + "Time"	Double
Allocation	Property	Seq. name + Allocation name	Boolean
	Class	"Allocations"	-
	Property	Device ID	Integer
Initial (pseudo step)	Property	Seq. name + "Initialized"	Boolean
Final (pseudo step)	Property	Seq. name + Final (step) name	Boolean
Fork (pseudo step)	Property	Seq. name + Fork name + "Branch" + #	Boolean
Join (pseudo step)	Property	Seq. name + Join name	Boolean

Steps, Allocations, (sub) Sequences and pseudo steps are in the algorithmic code handled with "if – else if" constructs so that they can be each entered only once [P4]. This is necessary because Modelica models are executed cyclically so that execution must continue from the phase in which it ended in the previous cycle. Steps keep a record on which Activities have been executed. Allocations are assumed to be next to

[13] The transformation creates variables for Transitions only if their transition condition is of type timeout.

initial (pseudo steps) and are released automatically at the end of the sequence. Fork-to-join regions are handled with a variable for each branch. For the execution to proceed from a Join, all the branches must have reached it. An example illustrating how algorithmic constructs are created to simulate a simple Sequence is presented in Figure 16.

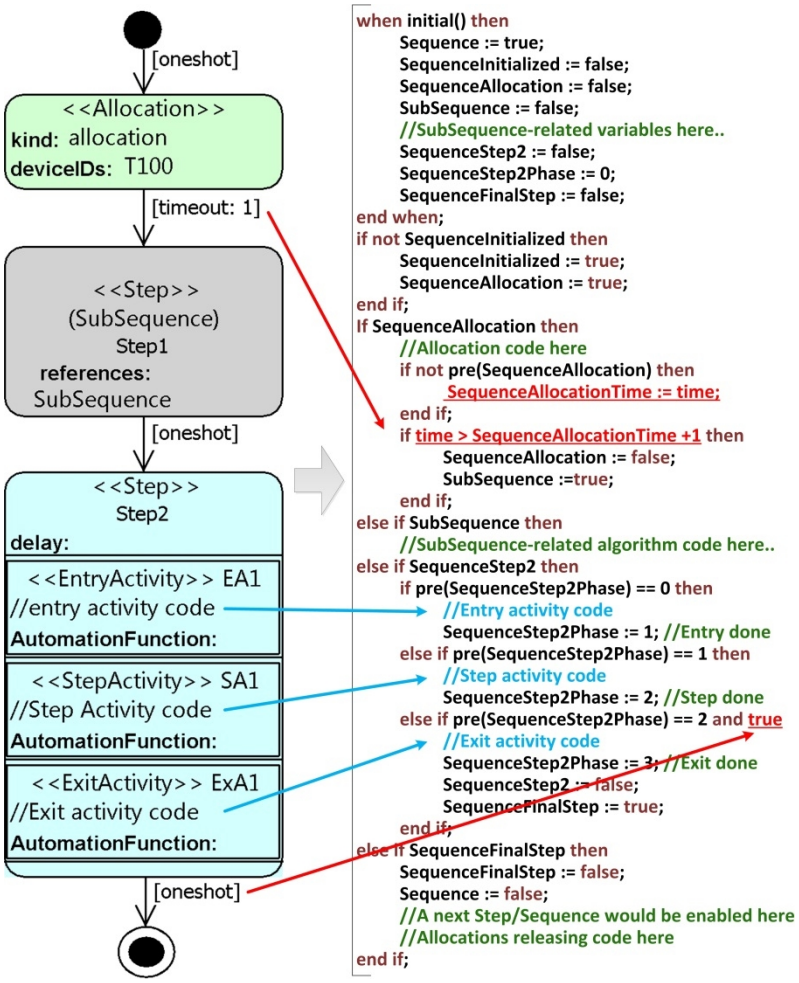

Figure 16 An Automation Sequence Diagram (ASD) and the corresponding Modelica algorithm section. (From [P4)

As presented in [P4], the Modelica code structures resemble the structures that can be used for executing UML state machines, see [81] and [82]. The ASD diagram type has also been extended from state machines [P4]. However, because of differences between the modeling notations, the work related to simulating state machines could not be re-used directly. For example, Sequences of UML AP can have several branches executing concurrently and independently of each other. Sequences can also include Allocations, for which there are no corresponding concepts in state machines. Another related

notation of UML, activity diagrams, would not enable activities to be broken up e.g. to StepActivities and ExitActivities [P4].

The restrictions of the developed simulation approach include that Sequences are always contained by AFs, so that AFs describe their sequential behavior. However, an AF can contain several Sequences that can be executed simultaneously. In addition, it is currently required that branches exiting a Fork in a Sequence meet each other in exactly one join. Lastly, a restriction in the approach is that it does not support looping so that Steps of a Sequence would be executed continuously, several times. However, in published simulation experiments, in which Sequences have been used to model e.g. start-up sequences and changing operation points, the restrictions have not caused difficulties.

4.3.3 Observations from Applying the Simulation Approach

As summarized in [P5], the simulation approach has been developed incrementally, in an agile manner and published in several articles including [P3] and [P4] as well as [83] and [84]. The basic transformation approach, as presented in [P3], has been specified with QVT operational mappings language. The executable (Java) transformation code that is used in the Eclipse environment was generated with SmartQVT tooling and extended with a custom Java class, in order to implement the interactive features of the transformation, for example. The transformation was then integrated into the tool environment by packaging it to a plug-in that implements an extension to the export extension point of the tool. (See Section 3.4.3.) The Logic Diagram modeling concepts and the diagram type, which was not part of the original UML AP tool, were implemented according to the procedures described in Sections 3.4.1 and 3.4.2, by extending the metamodel with EMF and by implementing the diagram types with the Topcased tooling.

In the simulation experiments that are presented in the articles ([P3], [P4], [83] and [84]) the plant models to be controlled have covered both machinery and process industry processes. In terms of numbers of equations, the complexity of the controlled processes has varied from a few equations to over 1400 equations, which also demonstrates the scalability of the approach to practical, non-trivial processes with industrial size and complexity [P4], [P5]. The control applications that have been simulated have covered different kinds of control functions including binary valued and feedback control, interlocks and sequences.

The observed benefits from applying the simulations are summarized in [P5]. The simulations have enabled prototyping interlocks and control solutions, comparing alternative interlock solutions as well as searching acceptable tunings for controllers to achieve sufficient dynamic performance. Simulations have revealed shortcomings in requirements and implementations. Lastly, simulations have been used to study exceptional and hazardous situations, for example by assessing interlock solutions during hazardous set-points. With respect to the common aspects of basic control systems, the following sub-sections summarize how they are supported by the simulation approach and in which publications the aspects have been addressed.

4.3.3.1. Binary and Feedback Control

Feedback and binary (valued) control structures, for actuators such as motors and valves, have been utilized in all the published simulation experiments. The simulated structures have been most often control loops consisting of measurement, control, output and possibly interlock AFs. In [3], a crane system was driven with three feedback-controlled (and interlocked) motors. In [4], the control solution for a pulp batch production system included 2 feedback controllers, one of which was binary valued. Binary valued controls were also used for several valves, according to a control sequence [P4]. In [83] and [84] the plant models were controlled with an interlocked feedback control loop and with three individual control loops, respectively.

Binary and feedback control loops can in the approach consist of both platform independent and platform specific AFs, for which simulation counterparts can be re-used from libraries. In addition, however, binary valued controls can be specified with Logic Diagrams, for example to activate or lock actuators in specific circumstances, and with Automation Sequence Diagrams, for example to activate or lock actuators according to progress of a control sequence.

4.3.3.2. Interlocks and Safety Functions

Interlocks are often specific to applications and difficult to re-use. Interlocks are used in control systems to protect the systems to be controlled from causing harm to themselves or humans [P4]. For example, they can be designed to stop devices and actuators or to constrain set-points based on the measured states of the systems. Currently, UML AP supports the specification of interlocks with Logic Diagrams that are used by the simulation transformation as a basis for creating new simulation classes. Support for interlocks was, thus, developed into the language after [P2] in which support for them was assessed as an important further development target. In the published simulation

experiments, interlocks were defined and simulated in [P3] to constrain the set-points for the trolley position and jib angle of a crane. In [P4], a Logic Diagram was used to specify a temperature controller (thermostat). For [83], Logic Diagrams were used to specify alternative interlock approaches for a cart system which were then simulated in order to compare their performance.

In addition to interlocks, Logic Diagrams could be used for the specification of logic of safety functions. However, in addition to the actual interlock (and safety function) logic, interlocks and safety functions often require locking and releasing outputs to actuators and devices. To enable this, platform specific AFs can be defined for actuators so that the AFs include interface ports for the signals.

4.3.3.3. Control Sequences

In addition to interlocks, also sequential control activities are often specific to applications. Sequences are needed by, for example, process industries to perform the start-ups of complex processes such as power plants and to drive the processes to their designed operation points. In a similar manner, shutting down a process in a controlled manner may require changing set-points as well as activating and disabling devices in a specific order [P4].

On the other hand, batch processes constitute a challenging domain of industrial processes. In batch industries, production processes can require, for example, the addition of source materials and substances according to the time constraints and achievement of defined process states such as temperatures and concentrations. In UML AP, sequential control activities can be defined with Sequences that are described with Automation Sequence Diagrams (ASDs) and enable an SFC conformant modeling notation. The execution of a single Sequence is centralized in an AF so that in simulations the contents created based on an ASD are placed into the simulation class that corresponds to the AF that owns the Sequence.

In the published simulation experiments, Sequences have been used only in [P4], to specify a control sequence for batch processing of pulp. However, as presented in [P4], the support for sequences could have been useful also in the other simulation experiments that had been published earlier. Sequences could have been used to define set-point trajectories to evaluate the controlled systems in different conditions. Without the support, the trajectories in e.g. [P3] and [83] needed to be defined manually.

4.4 Discussion

In MDD of control applications, simulations can follow MiL, SiL, PiL and HiL approaches. The simulations can use either a single simulation engine or co-simulation. However, because of the application domain specific characteristics, in MDD the focus should be on MiL simulations. Control applications are commonly developed for (PLC and DCS) control system platforms that already support the use of simulations. In order to obtain additional significant benefits, in MDD it should be possible to apply simulations earlier, before generating code. Practically this means using models of the control applications for (MiL) simulation purposes. The use of MiL simulations during control application development does not restrict the use of other (later) simulation approaches e.g. after hardware design. Instead, by selecting a suitable plant simulation language, the plant simulation model can be re-used.

In MDD in the domain, MiL simulations can be achieved by (co-)simulating the parts of the closed-loop system in different, connected simulation engines and by using model transformations. Model transformations can be used to, for example, transform control application models to plant models or vice versa. Of these approaches, co-simulation has been used e.g. in [71] and the approach to transform plant models in [68] and [69]. Included publications [P3] and [P4] describe the approach of the author to transform control application models to plant models using ModelicaML as the target simulation language. The important features of the approach include the ability to use simulation during both platform independent and platform specific development phases, so that simulation counterparts of platform specific blocks can be used.

In addition to the mentioned approaches, closed-loop simulations could have been developed by modeling both plant and control application models with a (the same) simulation language or by transforming both plant and control application models. However, these approaches would have restricted suitable modeling languages or required developing and keeping up-to-date two complex model transformations.

The co-simulation approach and the transformation based approaches to develop closed-loop MiL simulations were compared in [P5]. Based on the comparison, it is not possible to provide a clear conclusion on which of the approaches should be used. With the co-simulation approach, plant simulation models need not be processable with model transformations. However, technically it can be the most demanding approach and it can cause additional work with simulation cases. The approach also requires a simulatable control application modeling language or a transformation, so that the use of co-simulation does not always reduce the amount of required transformations.

The transformation based approaches do not require the coupling of simulation engines. However, when transforming plant models, the control application modeling language must be simulatable or an additional (second) transformation is needed. Without an additional transformation, the approach would not support, for example, UML as a control application modeling language. Similarly, transforming control application models to plant models requires the simulatability of the plant models or an additional (second) transformation. However, in the approach, languages such as UML can be used for control application modeling provided that it is possible to generate appropriate simulation counterparts for the models. In case of UML AP models, which are not simulatable as they are, the simulation model generation was possible.

The benefits of using simulations in control system development have been reported by several researchers and include, among others, improvements to design, development and validation of control programs and strategies. With simulations, it is possible to improve operator training and to test the control system in dangerous situations. As suggested in [P5], it is possible that many of the general benefits of simulations could be obtained also by using simulations in MDD context. With MDD techniques, it is possible to automate simple, repetitive tasks. However, the use of MDD does not necessarily affect the need to validate designs and decisions. Within MDD context, using MiL simulations, it could be, however, possible to simulate earlier because, for example, control system hardware design would not have to be completed.

The restrictions of MiL simulation within MDD context are related to missing hardware. Without a realistic UI, for example, the use of simulations to operator training could be difficult. However, there should be no reason why early MiL simulations could not be used for simulation tasks that do not require hardware as such. For example, verification and validation of control logic, strategies and tunings as well as prototyping and testing in small modules are possible [P5].

As summarized in [P5], the developed simulation approach has been used in the development of both machinery and process industry applications. In the published experiments ([P3], [P4], [83], [84]), the approach has enabled prototyping, experimenting and comparing control and interlock solutions, determining control tunings as well as detecting inconsistencies in requirements and designs. It has also been possible to study exceptional situations. Similar benefits have been reported in the other referred approaches to integrate simulations into MDD in the domain, for example the early validation of control applications [68] with reduced time and effort [69].

Yet to be addressed research questions for future research include how to select and connect simulated test cases to the MDD process. In order to facilitate simulation-assisted MDD, it should be possible to increase test coverage by selecting scenarios in a smart and systematic manner. At the same time, the process should support, for example, traceability between the modeled requirements being tested, the parts of design being tested and the test cases.

5 Safety in Model-Driven Development of Control Applications

The use of MDD techniques in safety system development has been suggested by few researchers. However, the modeling of safety aspects, perhaps in conjunction with a more traditional development process, has drawn more research attention. With techniques that are commonly used in MDD, e.g. UML and SysML, the safety aspects and their modeling have been addressed in several modeling profiles. Such profiles have been specified for different sub-domains of safety applications.

Work related to using model-based techniques in safety system development has been carried out in the DECOS project. The project is targeted to the development of both critical and non-critical functions of embedded control systems [85]. In the approach, the preferred means for specifying applications and their functionality is SCADE language (Safety Critical Application Development Environment) which is based on a formally defined data flow notation. It enables simulation at model level and code generation that has been certified against IEC 61508. [73]

In [86], Biehl et al. attempt to integrate safety analysis into the model-based development of embedded control applications in automotive industry. The objective of the work is to enable early safety analysis. The solution is to translate the concepts of the automotive domain to the generic concepts of safety and error analysis domain. In [87], Guillerm et al. discuss the use of SysML to address requirements specification, traceability as well as verification and validation in a model driven systems engineering process. In the paper, they extend SysML with a profile that supports, for example, the modeling of risks and requirements as well as traceability between them.

UML air-worthiness profile, see [88] and [89], extracts the key safety-related concepts of RTCA DO-178B standard into a UML profile in order to use them to facilitate communication between different stakeholders in software development. The standard, RTCA DO-178B, is the de-facto standard within the domain of commercial and military aerospace systems that contain software. UML safety analysis profile [90] documents hazards by presenting their occurrences with the use of Fault Tree Analysis (FTA) notation. The notation enables the modeling of condition sequences leading to hazards. Faults and hazards can be traced further to design in order to improve traceability. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoSFT) [55], lastly, includes notations to model risk assessments. Special attention is paid to the description of the hazards, risks and treatments of the risks.

Architecture design is an essential part of safety system development. Software architecture is a fundamental organization of a software system as embodied in its components, relationships between them and to the environment, and the principles guiding its design and evolution [91]. For safety systems, architectures and architectural solutions are addressed by several standards. For example, IEC 61508 [1] gives guidance for selecting architectural approaches related to fault and error detection and handling, timing as well as management of resources, among other aspects. Redundancy and redundancy-related design patterns, so that a calculation is performed redundantly or observed by another channel to produce a reliable result, are presented both in standards and related literature. For example, redundancy solutions are described in the 6th part of IEC 61508. Publication [92] presents the design patterns of safety systems including: Homogeneous Redundancy, Diverse Redundancy, Monitor-Actuator and Safety Executive patterns. Design patterns that are targeted to basic control and safety systems as well as cooperation between them have been presented also in [93] and [94].

In addition to architectural patterns, another important aspect related to software architectures is the utilization of architectural knowledge during the development of a system. Architectural knowledge can be defined as a sum of architectural design and architectural decisions, including rationale for the design [95] . Architecture Knowledge Management (AKM), on the other hand, includes coordination and management of artefacts (e.g. requirements, design patterns and decisions) related to the architecture of a system.

This Chapter discusses extending the information content of models, which are used in MDD, with safety aspects. Requirements and objectives for the extensions are specified in Section 5.1. Possible modeling notations and techniques are discussed in Section 5.2. The developed extensions as well as tool support to generate documentation based on the extensions are presented in Section 5.3.

5.1 Requirements for Modeling Safety Features

Related to safety systems and applications, this thesis focuses on extending the information content of models that are used in MDD with aspects and characteristics that are required for safety systems. In addition to critical safety systems, however, they can be of importance also for basic control systems. The techniques and solutions that improve safety and related quality attributes in safety systems improve them also in basic control systems [P8]. Generally beneficial quality attributes that should be taken

into account in safety and basic control system development alike include, among others, maintainability, security and reliability.

Safety is also a virtue that should be a priority in the development of any control system - including those of processes and plants that are not capable of causing significant harm and are controlled by basic control systems only [P8]. On the other hand, even in case of potentially hazardous processes that have specific safety systems, the safety related functions of basic control systems, e.g. interlocks, can be beneficial. Such functions can be developed to treat hazards and their risks and thus constitute non-certified treatments to the risks to improve the overall integrity. They could be seen as additional, non-certified safety barriers [96] or layers of protection [97] in the defense-in-depth principle. Functions of basic control systems, however, are not usually critical and their operation principles may differ from those of critical safety functions. Whereas the (sole) purpose of safety functions is to maintain safety, the safety related functions of basic control systems can be related to productivity, too. Before hazardous limits, it is also appropriate to apply different, more complex approaches to react to the deviations.

Before critical situations, it can be feasible to try to recover from deviation situations to maintain productivity. At critical limits, safety functions are often designed to perform a controlled shut-down of the plant or process and to de-energize hazardous devices, if possible. The shut-down-approach is with many processes both a simple and effective approach to achieve a safe state. The downside of the approach is the lost productivity of the process, plant or machine. Restoring productivity after a shut-down may also require the complex manual operations of the operating personnel. However, in basic control systems, the approaches to recover from the deviations can be more advanced than in safety systems. There is no need to develop basic control systems according to safety standards or to certify them. Especially, before the critical limits, it is not necessary to apply a simple approach to guarantee safety.

In addition to sharing a process to be controlled, basic control system development can benefit from information that originates from safety system development. Whereas hazards and their associated risks are the basis for developing safety systems, they can aid understandability also in case of basic control systems. Hazards and their related risks can provide a rationale for the requirements of safety related functions of basic control systems [P6]. Identified hazards can point out the ways in which it is possible for the systems to cause harm to their environment. Including this information in models can improve the awareness of control system developers over the hazards and related risks of the controlled systems.

The question, whether safety systems could be produced with model-driven techniques in future is interesting. Answering the question thoroughly is out of the scope of the thesis. Nevertheless, to develop safety functions with MDD techniques, one would have to be able to meet the requirements of safety standards and to produce the required documentation. Preferably, the documentation should be produced without excessive manual work, with MDD techniques. To enable this, the required information would need to be in the models. The following sub-sections summarize requirements related to the modeling of hazards, risks and use of standard solutions as well as to supporting traceability, correctness and completeness in models.

5.1.1 Hazard and Risk Information

The development process of safety systems, considering the requirements of safety standards, is risk driven. For example, IEC 61508 [1] is a standard with a risk driven development process. After scope definition, the development process starts from the identification of hazards and determination of risks. They are followed by the specification and allocation of requirements (to treat the risks) and then proceeding towards implementations [P6]. The phases of the process build on information produced by earlier phases, starting from the hazards. To the different development phases, standards suggest development techniques and measures that promote the properties of systematic integrity such as correctness and completeness. Traceability should ascertain that the hazards are the basis for – and become treated by - the safety functions [P6].

Hazards and their associated risks are thus the basis for specifying and understanding safety requirements. However, hazards should be in the scope of models also for, for example, traceability purposes [P6]. The hazards of the processes to be controlled should be visible to developers that use the models in MDD. Individual hazards should have identities with which they can be referred to and linked to other engineering artefacts in models and in the development process. Risks, which are associated to hazards, should be included in the models. They present the significance of the hazards with respect to their likelihood and consequences.

It should be possible to describe, in a structured manner, how hazards can occur. This could improve understanding about the hazards themselves and what is required for them to be realized. The detailed information on their realization could also facilitate the specification of the requirements for the safety (and safety related) functions, and help understanding how the functions can prevent the hazardous situations from occurring.

5.1.2 Traceability, Correctness and Completeness

In safety standards, e.g. IEC 61508 [1], a repeating requirement for the phase products of development is traceability between them [P6]. For example, safety system requirements need to be traceable to both perceived safety needs (hazards) and software safety requirements [P6]. Similarly, software safety requirements need to be traceable to both design elements that implement the requirements and to test cases evaluating their fulfilment. In addition to traceability, repeating requirements for phase products are correctness and completeness [P6]. Correctness and completeness are also properties of systematic integrity based on which IEC 61508 [1] (in the third part of it) recommends many techniques to be used and not to be used in different phases of safety system development.

Based on the traceability information, it should be possible to confirm, for example, that all hazards and requirements are addressed (completeness) by the design of a system or by some part of the design. In addition to performing this (and similar) consistency checks, the (traceability) information is required for several other purposes, e.g. for focusing inspections on correct parts of design. In addition to functional requirements, models should address other safety related requirements and their traceability. Similarly to functional ones, these requirements – imposed by standards and regulations, for example - should be traceable to design artefacts that fulfill them.

To support correctness, models and diagrams should be intuitive and on appropriate abstraction levels. It should be possible to use models with preferably domain specific or otherwise intuitive and informative diagram types. For example, requirements should be specified in a formal enough, unambiguous manner and be based on concepts that are understandable for the developers. However, the current support of UML for requirements specification is limited and based on the use case concept, mainly [P6].

5.1.3 Use of Standard Solutions

The development process of safety systems and applications, including solutions, techniques and measures to be used during the development is governed by standards. A developer of a software part of a safety system should apply standard compliant techniques, measures and solutions. However, in addition to using them, a developer of such a system must be able to prove the compliance of the system. This is where appropriate documentation is needed [P8]. In order to produce the required documentation from models with MDD techniques, the information should be in the models and it should be possible to be gathered to a suitable form. For example, it could

be useful to enable generating (gathering) documentation on whether or not - and which - recommended techniques have been used in design and whether or not the techniques are appropriate for the safety levels required from the applications.

It is also possible that the strict documentation requirements of safety systems are a reason for the scarce use of MDD in safety system development. As discussed in [P8], the reason is not that standards would not allow the use of MDD approaches that use suitable modeling techniques. Instead, for example IEC 61508 recommends automatic software generation as an architecture design technique for all safety integrity levels. However, in general, models tend to be more applicable to representing solutions than the rationale behind them. For example, many basic concepts of UML are similar to the concepts of object-oriented programming languages and can be in MDD used as a basis for code generation. However, information on why something has been designed in the way it has, or that a solution is a standard one, is often missing. Given the strict documentation requirements, it is possible that MDD has not seen to offer possibilities to improve the efficiency of development [P8].

5.2 Considerations on Implementation Techniques

5.2.1 Modeling of Hazards and Risks

The reasons to include hazards in scope of modeling are various, including the ability to support traceability between them and requirements as well as to improve the understandability of requirements. The modeling of hazards, however, is not supported by standard UML or SysML. Nevertheless, there are well-known approaches to model the occurrences of hazards that could be taken advantage of. Such approaches include those of the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoSFT) of OMG [55] as well as the safety analysis profile [90]. In addition, the profile in [98] supports the modeling of hazards with FTA models and with Failure Mode, Effects, and Criticality Analysis (FMECA) models. However, the work is focused on incorporating safety requirements in software architectures and evaluating the architectures.

In QoSFT [55], the main objective is not to enable the specification of how hazards can be realized. Instead, attention is paid on documenting the magnitudes of risks, i.e. their likelihoods and consequences, as well as the compromised assets, stakeholders and treatments of the risks. Approaches to treat the risks include avoidance, the reduction of

likelihood or consequences as well as retaining and transferring the risks. The tracing of risks to requirements is not covered in the profile.

The safety analysis profile [90] covers both the occurrences of hazards and tracing of them to requirements. The definition of how hazards can be realized is supported with FTA modeling. FTA models can be used also in a quantitative way and they can aid the development of safety functions. Safety functions can be designed to stop the fault and event sequences that lead to hazards, for example by shutting down hazardous devices, so that for the hazards to occur, also the safety functions would have to fail [90], [P6].

5.2.2 Requirements and Traceability

Traceability is a property the achievement of which can be challenging with traditional, document based development processes. Traceability between identified hazards and requirements, for example, can be supported in a simple case by specifying explicitly the unique identifiers of hazards that the requirements have been specified to treat. However, references between documents and specifications can be difficult to keep up to date when something is changed. In addition, the generation of traceability matrices (or other summaries) from the traceability information and searching for possible inconsistencies could be difficult to automate [P6]. By including the information in models, some of these tasks could be at least facilitated with automated functions.

However, support for traceability is limited in UML [P6]. A UML profile, SysML, covers traceability with specific relation types such as satisfy and verify. Satisfy relations can be used between SysML requirements and design artefacts that fulfil them. Verify relations can be used between requirements and (SysML) test cases determining their fulfilment. Relations of SysML can be also searched from models in order to, for example, generate tables or matrices. The SysML traceability concepts do not support all the traceability requirements of safety standards but form a basis that can be extended [P6]. Extensions have been specified, for example, in the safety analysis profile [90] that supports the tracing of hazards to requirements. In addition, the tracing of requirements to design was supported with trace relations already in the original UML AP specification [52].

In addition to traceability, support for requirements specification is limited in UML [P6]. Of the concepts of UML, only use cases are intended for specifying interactions between systems and their users. However, because of limited information content of (UML models of) use cases, they are often accompanied with separate descriptions (documents) on what is required to happen. In SysML, requirements can be specified

with textual requirement concepts that give requirements exact identities but can hardly be characterized as formal. UML AP requirement concepts extend SysML ones with a classification based on the basic need and could also be extended with safety related information content. With safety related information, they could also enable automating various consistency checks. For example, [P6] mentions checking the compliance of safety (integrity) levels of requirements that are associated to each other and checking that approaches to reduce risks are documented in the models.

Domain specific requirement specification techniques, which are not related to UML, include IEC standards 62424 [53] and IEC 61804 [99]. These standards cover the structured presentations of required control functionality and may be familiar to professionals in the domain [P6]. In addition, logic diagrams have been traditionally used in the domain for the specification of, for example, safety related interlocks [100]. As a semi-formal technique, logic diagrams are acceptable for requirements specification, in addition to detailed design and architecture design, also from the point of view of the safety standards.

IEC 62424 [53], which is also a supported source information format in the AUKOTON development process, defines a specification for the representation of Process Control Engineering (PCE) requests. PCE requests can be used in Piping and Instrumentation (P&I) diagrams and they enable data exchange between P&I tools and control engineering tools. IEC 62424 also allows the identification of PCE requests that are related to safety and annotating their respective safety levels. The levels can be categorized with SILs or Performance Levels (PLs) of EN ISO 13849-1 [101]. However, defining precise safety function logic is not supported in IEC 62424.

IEC 61804 [99], on the other hand, originates from the power generation industrial sector and utilizes IEC 61499 FBs for detailed requirements specification. Before using the FB language, required control functions are identified and marked in P&I diagrams and structured for presentation in control hierarchy diagrams.

5.2.3 Standard Solutions in Models

One of the key concepts of MDD is the shifting of development efforts from (written) documents to models and the ability to automate parts of the model processing. For special purposes, e.g. safety system development, it could be possible to maintain separate documentations. However, this would be against the idea (of MDD) and could reduce the potential to benefit from MDD. A more appropriate approach would be to include the required information (for producing the documentation) in the models in the

first place [P8]. The information on (also) the use of standard solutions and techniques should, thus, be in the models. Models should include information on where and how specific techniques and solutions have been used.

However, as discussed earlier in Chapter 3, modeling languages such as UML include weak support for standard solutions and, for example, design patterns. Also in general, models tend to present rather solutions than rationale behind them [P8]. In another publication [102], these challenges are addressed with the means of Architecture Knowledge Management (AKM) and use of an Application Lifecycle Management (ALM) tool, Polarion[14]. Use of an external tool (in addition to MDD tools), however, could lead to redundant information [P8]. This is why some documentation functionalities presented in [102] have been in [P8] implemented by extending the design pattern modeling concepts. In this way, the functionalities have been included in the UML AP tool that is used throughout the MDD process.

Design patterns, with safety related extensions, could be used for documenting the use of standard solutions in safety system development [P8]. For example, extensions to the pattern concepts could be related to specifying the acceptability of the patterns for different safety levels. Design patterns could then be used for modeling and marking uses of techniques that standards recommend. For example, in IEC 61508 [1] development techniques, measures and solutions can be recommended (R), highly recommended (HR), non-recommended (NR) or without a recommendation for each SIL [P8]. For a system of a specific safety level (SIL) a developer should use recommended and highly recommended techniques and avoid the use of non-recommended ones.

5.3 Safety Related Extensions to UML AP

5.3.1 Hazard and Risk Information

The hazard modeling approach that has been integrated into UML AP is presented in [P6] and utilizes the well-known Fault Tree Analysis (FTA) notation. FTA modeling is also used in both [90] and [98]. FTA diagrams allow depicting the occurrences of hazards in a graphical manner. They are analytic and intuitive for both safety system and control system developers. Occurrences of hazards can be presented as the logical

[14] http://www.polarion.com/products/alm/index.php

combinations and sequences of faults and other conditions. FTA models could be used also for the quantitative analysis of hazards, for example by including probabilities of faults and conditions in the models and by applying probability calculus [P6].

FTA is supported with a set of modeling concepts that enable depicting fault and condition sequences that can lead to hazards. The concepts have been developed to extend the UML AP implementation and they are presented in Figure 17, which is based on pictures and description in [P6]. Hazards as well as fault and condition sequences leading to them are presented with different types of HazardModelNodes. The node types include Hazard, RequiredCondition, ResultingCondition, Fault as well as LogicalOperations, which are not shown in the figure. In the approach, Hazards include attributes for the specification of their associated risk values so that risks cannot be modeled independently from hazards. In this way, it can be easily made sure that each Hazard in a model has an associated risk value, so that models are not incomplete with respect to this aspect.

The tracing of hazards to requirements is supported in the approach with the RiskTreatment concept. In addition to supporting traceability, RiskTreatments enable documenting the approaches to treat the risks. The treatment options, which are based on the alternatives in QoSFT profile [55], are in the metamodel specified in the RiskReductionApproach enumeration. Hazards can be related to each other with different kinds of relations that are classified in the HazardRelationKind enumeration. Additionally, the metamodel defines two other enumerations: LikelihoodKind and ConsequenceKind. The classification of risks that is supported by the concepts follows the qualitative classification in annex B of part 5 of IEC 61508 [1]. With agreed limits, a quantitative classification or some other qualitative classification could be supported as well.

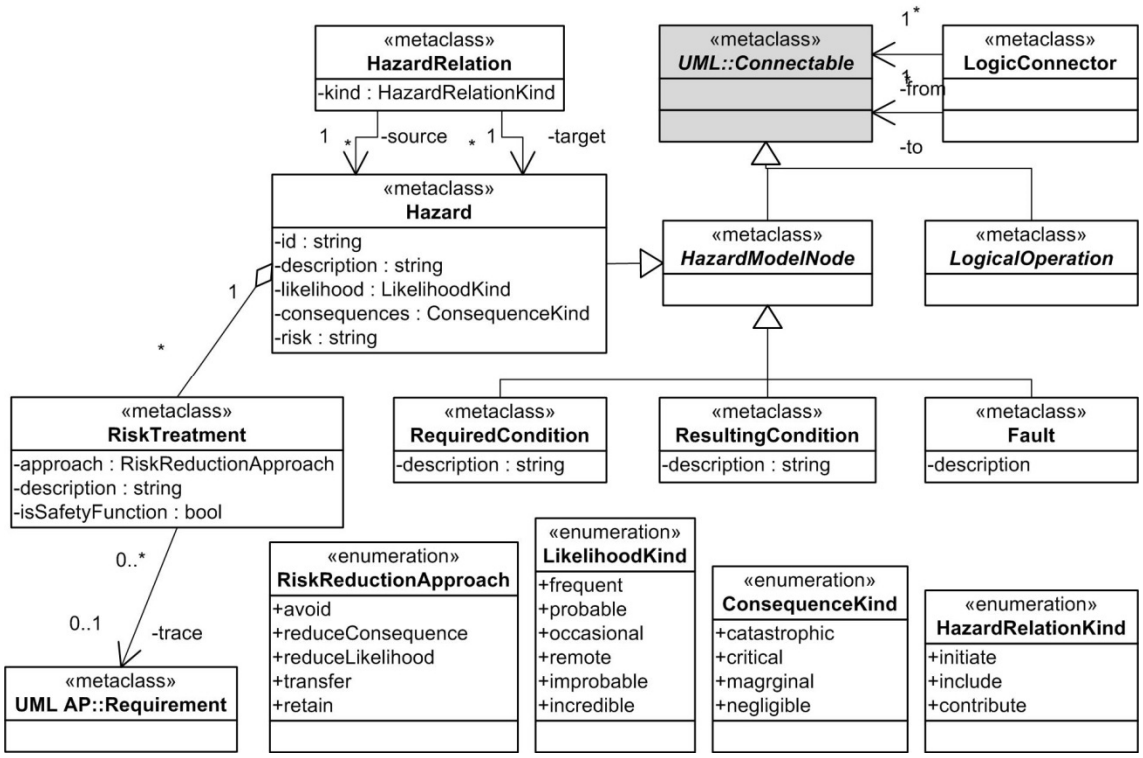

Figure 17 Metamodel of the (FTA) modeling concepts excluding concrete logical operation types. (The metamodel is based on the pictures and description in [P6].)

Thus, instead of a new one, an existing, well-known notation (FTA) was integrated into UML AP to be used in the MDD context. This is also in general the approach by the use of which MDD could be extended for the needs of safety system development. Instead of new (unfamiliar) notations, already recommended ones could be used but in a new (MDD) context. In this way, it would be easier for authorities to allow the use of model-driven techniques in safety system development. However, in the MDD context, it could still be possible to benefit from the possibilities to automate model processing.

As suggested in [90], the use of FTA diagrams can also facilitate the development of corrective functions. The functions can often be designed to stop the fault and condition sequences that lead to hazardous situations. In this way, for the original hazard to occur, also the safety functions would need to fail in preventing the required conditions (of hazards) to be realized [P6]. To illustrate the simple idea, consider, for example, the simple FTA diagram in Figure 18. In the example tank system of [P6], running pump M100 dry can be caused by using the pump when tank B100 is empty. A simple approach to prevent the hazard from occurring could be to prevent the required conditions from being realized at the same time. For example, this could be achieved

with an interlock that would force the pump to shut-down when measured level in the tank is below an agreed limit.

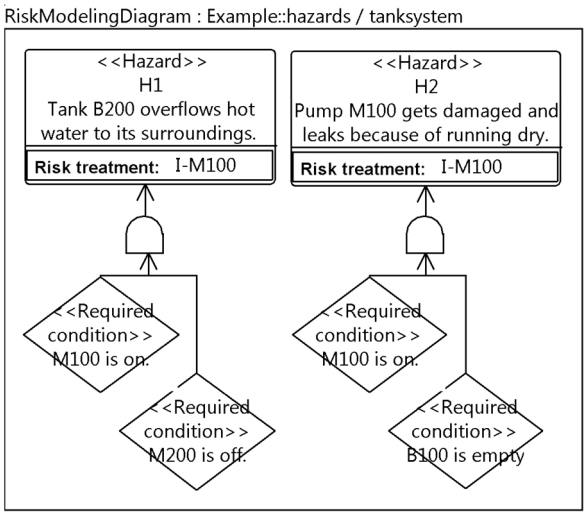

Figure 18 An example FTA model related to a tank system used as an example in [P6].

The FTA-related extensions to the UML AP modeling concepts and diagrams have been implemented with EMF and Topcased according to the procedures described in Sections 3.4.1 and 3.4.2. Tool supported functions that utilize the hazard concepts to automate documentation generation and to perform consistency checks will be presented in following sub-sections.

5.3.2 Requirements Modeling

Functional (UML AP) Automation Requirements are structured concepts with specific attributes e.g. for id and source. In a manner similar to AFs, they have been divided into a hierarchy, based on the basic need, for example to measure a quantity or to compute a control signal [P6]. To model dependencies between required functionalities the approach includes RequirementInterfaces. With RequirementInterfaces, it is possible to model, for example, a required control functionality being dependent on a required measurement functionality, i.e. that a measurement is needed by a control task. RequirementRefinements enable including additional information in Requirements. They include both a semantic meaning and value, to support various (possibly in-house) practices and needs. For example, a refinement could be used to define a measurement range related to an analog valued measurement. Two specific RequirementRefinement types are also used for including safety-related information in requirements. They are presented in Figure 19.

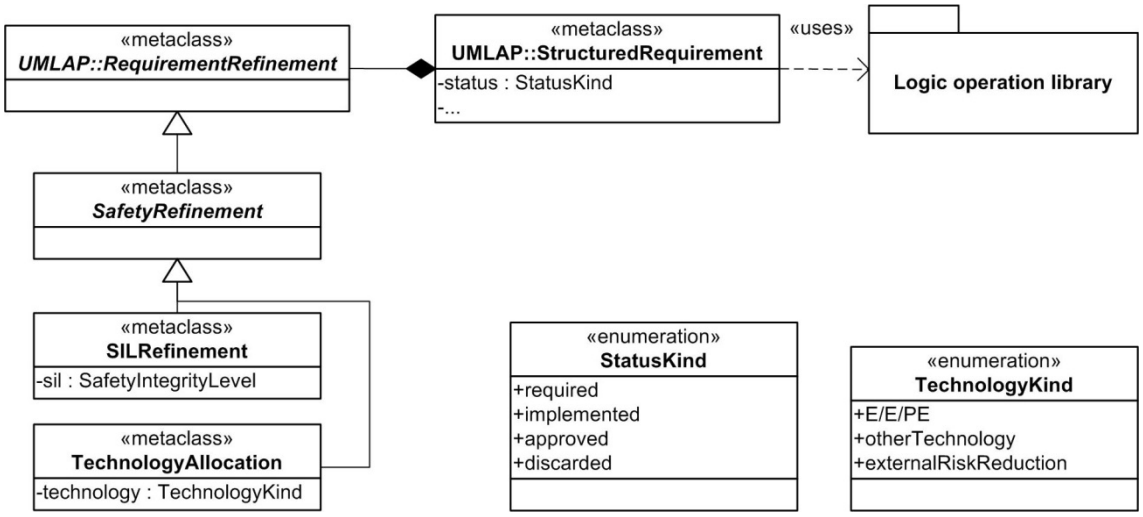

Figure 19 The safety related Refinements of UML AP that can refine StructuredRequirements. ((Modified from [P6])

With the first safety related refinement type, TechnologyAllocation, it is possible to allocate system safety requirements for E/E/PE (Electrical, Electronic or Programmable Electronic) safety systems, for other technology safety systems and for external risk reduction facilities. This classification is based on the classification in IEC 61508 [1] (part 1) that is used in the safety requirements allocation lifecycle phase. The other new refinement, SILRefinement, enables specifying required safety levels (SILs) for functional safety requirements. That is, SILRefinements refine functional requirements with information on which integrity levels the requirements must be implemented.

The detailed logic of safety and, for example, interlock requirements can be specified with UML AP Logic Diagrams that were discussed in Chapter 4 in relation to generating simulation classes. Logic diagrams, as a semi-formal method, are highly recommended by IEC 61508 for requirements specification on all SILs [P6]. They enable the specification of logic from input RequirementInterfaces to output RequirementInterfaces of Requirements. Technically, the use of UML AP Logic Diagrams (for this purpose) is in the tool environment possible because the root elements of the diagram type are UML Classes[15].

[15] The StructuredRequirement concept extends the SysML Requirement concept that in turn extends the Class concept of UML. Similarly, RequirementInterfaces can be used as Ports in Logic Diagrams because they extend the UML Port concept.

5.3.3 Traceability and Documentation Support

As presented, UML AP concepts include support for explicit traces between hazards and requirements that have been specified to treat the hazards as well as between requirements and implementing design elements. Between hazards and requirements, the trace concepts are called RiskTreatments. Between requirements and design (elements) the concepts are called TraceRelations [P6]. Technically the traces are one-directional. However, by iterating through traces in a model, it is possible to support backward traceability. For example, by iterating through RiskTreatments in a model it is possible to find all hazards that are traced to a requirement.

A RiskTreatment connects exactly one Hazard to exactly one (UML AP) Requirement. However, it is possible for a Hazard to contain several RiskTreatments. On the other hand, for a Requirement, there can be several RiskTreatments that trace different hazards to the Requirement. In this way, the treatment concept has been kept simple although it is still possible to e.g. specify that a Hazard is treated with several safety function requirements and that a Requirement (of a safety function) is to treat several Hazards. TraceRelations are contained by Requirements (that they trace), in a manner similar to RiskTreatments that are contained by the Hazards that they trace. A TraceRelation connects a Requirement with a number of implementing elements that take part in implementing the Requirement [P6].

Both RiskTreatments and TraceRelations are processable with, for example, model query and transformation techniques and they support both forward and backward traceability. To demonstrate how to benefit from the hazard, risk and traceability elements, they are used in the tool environment in several functions that have been developed to facilitate the development work, see [P6] and [P8]. Properties of systematic integrity that the functions have been developed to improve include correctness and completeness.

Included publication [P6] introduces traceability matrices that are compiled based on the traceability elements. The matrices are compiled (to MS Excel sheets) by collecting Hazards (or Requirements) to rows and Requirements (or elements) that they are traced to in columns. In addition to presenting the traceability information in a compact form, the matrices support completeness. Possibly overlooked hazards and requirements, which are not traced further but should be addressed in the design, are in the matrices highlighted (warned) with a red color.

An example hazard traceability matrix that was used in [P6] is presented in Figure 20. In the example below, all Hazards are traced further so that they are not warned (with the red color) by the tool [P6]. Such a Hazard (or a Requirement) that would not be traced further would indicate either a problem in the design or the model not being up-to-date. In either case, the situation should be warned by the tool so that it could be inspected by a designer. The traceability matrices were in [P6] exported to Microsoft Excel sheets. This documentation generation functionality has been later extended with documentation support related to design patterns in general [P7] and especially the design patterns of safety systems P8].

	A	B	C	D	E
1	**Hazard trace table**	InterlockingRequirement: I-M100	InterlockingRequirement: I-M200	InterlockingRequirement: I-Y101	
2	Hazard: H1	X			
3	Hazard: H2	X			
4	Hazard: H4		X		
5	Hazard: H3		X	X	
6					

Figure 20 An example Hazard traceability matrix from [P6].

In addition to traceability and completeness, including safety information in models can enable automating various consistency checks. As suggested in [P6], it could be easily checked that safety requirements that are related to each other have compatible integrity (SIL) refinements. For example, a required measurement that a required safety function is dependent on must have an equal or higher integrity than that of the (requiring) safety function. Otherwise, the safety function would become dependent on a measurement with lower integrity. In a similar manner, it could be easily checked that all Requirements are traced to (SysML) test cases and that all RiskTreatments document a risk reduction approach, for example.

5.3.4 Patterns of Safety Systems

The design pattern approach presented in [P7] and Section 3.4.4 is in [P8] extended for the modeling needs of safety systems. As summarized in [P8], the work is intended to facilitate the development of safety systems by supporting traceability between standard

(safety) solutions and their use in system designs, by enabling verification of safety levels of solutions (in comparison to required levels) and by guiding the selection of techniques and solutions.

A metamodel presenting both the safety-related and previous pattern modeling concepts is presented in Figure 21. In the metamodel, concepts that are additional to the previous pattern concepts ([P7]) are highlighted with a gray color. A central new concept in the metamodel is SafetyPattern, which is intended for modeling the design patterns of safety systems. The applicability of SafetyPatterns can be specified for different safety integrity levels [P8]. Similarly to recommended techniques and measures in IEC 61508 [1], a SafetyPattern can be Recommended (R), Highly Recommended (HR), Non-Recommended (NR) or with a Non-Specified recommendation (NS) for each SIL [P8]. The alternatives in the Recommendation enumeration in the metamodel correspond to these alternatives. SafetyPatterns, thus, enable modeling measures, techniques and solutions that standards recommend. However, the concept can be used also with patterns that are related to safety but for which recommendations are not available in standards.

SafetyPatterns that are related to each other can be collected into collections with the SafetyCatalogue concept. SafetyCatalogues are intended to contain patterns that are often used together and to which sets of patterns that are used in models can be compared. In a catalogue, patterns can be related to, for example, a development phase or a specific aspect in design. For example, IEC 61508 [1] includes various lists of techniques and measures to be used during different software safety lifecycle phases. For software architecture design, for example, the standard mentions 27 techniques, some of which are alternatives to each other or non-recommended for specific safety levels. [P8]

In addition to PatternRelations, between SafetyPatterns it is possible to use Specialization relations. The background of the concept is that for many architectural solutions that safety standards recommend, for instance, there are already published, more detailed patterns in literature. With the (Specialization) relation, more specialized patterns of, for example, pattern literature can be marked as specializations of the SafetyPatterns that the standards recommend. The specialized patterns can then be considered as alternatives for the general patterns, for example when comparing the sets of patterns (used in models) to SafetyCatalogues.

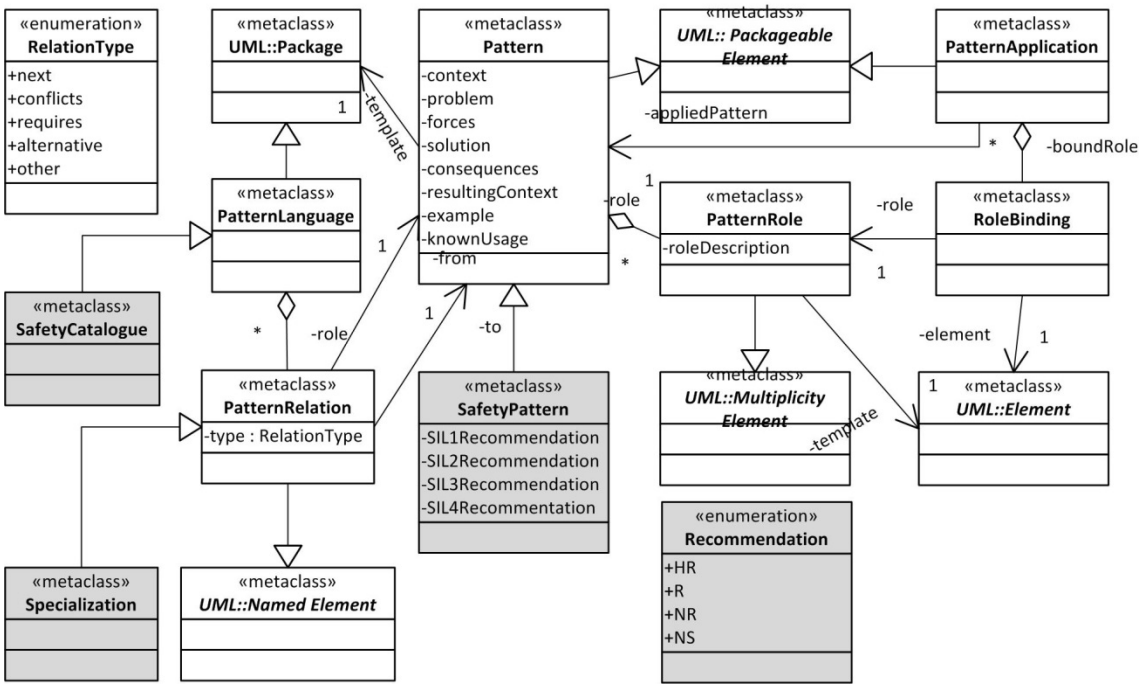

Figure 21 The metamodel of the SafetyPattern modeling concepts. (Modified from [P8])

The safety-related pattern modeling concepts enable automating various consistency checks, providing recommendations on the use of patterns and improving traceability in models. These capabilities are demonstrated in [P8] by extending the documentation generation support that has been briefly discussed in Sections 3.4.4 and 5.3.3. The new documentation sheets are described in [P8] and they are called Safety Catalogue sheet, Safety Catalogue Conformability sheet and Safety Pattern Traceability sheet.

Safety Catalogue sheets, firstly, enable presenting SafetyCatalogues in a tabular form that is similar to the form of the recommendation tables of IEC 61508 [1] (in annex A of part three of the standard) [P8]. On one hand, the sheet type is intended to facilitate the development of SafetyCatalogues to correspond to recommendations of standards. On the other hand, the tabular form can be used during development work when considering, for example, which solutions, techniques and measures to use. In the sheets, patterns are represented as rows of a table. Separate catalogues are printed to separate tables. The order of SafetyPatterns in a table is determined based on relations (next) between the patterns. Alternatives are in the tables assigned same numbers but different letters to indicate them being alternatives to each other [P8].

An example Safety Catalogue sheet from [P8] can be seen in Figure 22. The table presents 15 techniques and measures that IEC 61508 [1] recommends for software architecture design. Many of the techniques and measures are such that pattern literature

already includes specialized versions of them, for example to implement redundancy [28] or to recover from faults [29].

Safety Catalogue: "IEC 61508 Architecture Design"

#	Pattern	SIL 1	SIL 2	SIL 3	SIL 4
1	Fault detection	NS	R	HR	HR
2	Error detecting codes	R	R	R	HR
3a	Failure assertion programming	R	R	R	HR
3b	Diverse monitor techniques (with independence)	NS	R	R	NS
3c	Diverse monitor techniques (with separation)	NS	R	R	HR
3d	Diverse redundancy	NS	NS	NS	R
3e	Functionally diverse redundancy	NS	NS	R	HR
3f	Backward recovery	R	R	NS	NR
3g	Stateless software design	NS	NS	R	HR
4a	Re-try fault recovery mechanism	R	R	NS	NS
4b	Graceful degradation	R	R	HR	HR
5	Artificial intelligence - fault correction	NS	NR	NR	NR
6	Dynamic reconfiguration	NS	NR	NR	NR
7	Modular approach	HR	HR	HR	HR
8	Use of trusted/verified software elements (if available)	R	HR	HR	HR
	...				

Figure 22 A Safety Catalogue sheet example presenting 15 techniques and measures that IEC 61508 recommends for architecture design. (Modified from [P8])

Safety Catalogue Conformability sheets present SafetyCatalogues, SafetyPatterns of which can be found in the model from which the documentation is exported. Patterns of the catalogue, to which the model is compared, are presented in rows, in a manner similar to the Safety Catalogue sheet. A separate table is printed for each catalogue. SafetyPatterns, instances of which can be found in the model, are marked with a gray color to indicate traceability. In addition, the table supports correctness by highlighting (with color coding) whether the patterns that are used in the model would be appropriate for each SIL. Incompatibility can result from both use of a non-recommended SafetyPattern or not using a recommended (or highly recommended) SafetyPattern or any of its recommended alternatives [P8]. The last rows of the table present the numbers of patterns (excluding alternatives) that have been used and that would be recommended for each SIL.

An example Safety Pattern Comformability sheet from [P8] is presented in Figure 23. The sheet presents how requirement specification techniques (that are used in an example model) conform to a SafetyCatalogue that has been modeled to correspond to the recommendations of IEC 61508 for software requirements specification. According to the table, the model includes markings of use of semi-formal methods, forward and backward traceability as well as use of computer-aided specification tools. According to the table, they are recommended for all SILs and it would not be necessary to use other techniques and measures in order to conform to the catalogue.

Conformability to "IEC 61508 Software safety requirements specification"

#	Pattern	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	R	R	HR	HR
1b	Formal methods	NS	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR
	Pattern usage:	4/4	4/4	4/4	4/4
	Pattern usage (%):	100.0	100.0	100.0	100.0

Figure 23 A Safety Catalogue Conformibility sheet presenting the usage of requirements specification techniques in a model and their conformability to the recommendations of IEC 61508. (Modified from [P8])

With the information on required safety levels, on traceability from requirements to implementations and on solutions used in the implementations, it is also possible to check the consistency between them. This is automated in the third new sheet, SafetyPattern Traceability sheet. The sheet traces safety-related requirements (that include SILRefinements in them) to Packages that contain implementing design elements for the requirements and to SafetyPatterns that are used in the Packages [P8]. The table presents SILs related to the requirements, integrity levels required from the Packages (which are derived from the requirements) as well as recommendations of the Patterns for each SIL. Uses of recommended and highly recommended patterns are indicated with a green color whereas uses of non-recommended patterns are warned with a red color.

An example Safety Pattern Traceability sheet from [P8] is presented in Figure 24. According to the sheet, the model of the example contains two (safety-related) requirements of safety level SIL 1. The requirements have been traced to elements contained by the Software Safety Requirements and ControlStructures Packages, so that the safety levels required from the Packages are the same. Lastly, the sheet presents the SafetyPatterns, which are used in the Packages in question, and indicates, with a green color, that the patterns are recommended for the safety levels in question.

Traceability: Requirements --> Packages --> (Safety)Patterns

REQUIREMENT:							
P100 Protection (SIL1)	↘						
P100IR (SIL1)		↘					
PACKAGE:	Software Safety Requirements (SIL1)	ControlStructures (SIL1)	**PATTERN:**	**SIL1**	**SIL2**	**SIL3**	**SIL4**
		↘	Automatic software generation	R	R	R	R
	↘		Semi-formal methods	R	R	HR	HR
	↘		Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR
	↘		Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR
	↘		Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR

Figure 24 A Safety Pattern Traceability sheet presenting the traceability of the safety requirements of an example system to implementing Packages and to SafetyPatterns used in the Packages.

In addition to the presented sheets, [P8] envisions tool-supported functions that could be used in a constructive manner. For example, a guided process could start from modeled requirements that would determine required SILs for required functions. A developer could then select a SafetyCatalogue to be used to guide the design or a specific design phase, for example architecture design. Based on the catalogue selection and required SILs, the tool could suggest patterns to be used. In practice, this scenario could be implemented as simply as a modification to the Safety Catalogue sheet, by hiding inappropriate patterns based on required integrity levels [P8].

The new modeling concepts that are required for SafetyPatterns and for the sheets, see the metamodel in Figure 21, have been implemented by extending the EMF metamodel used by UML AP tool according to the procedure described in Section 3.4.1. The sheets, which present safety-related information, on the other hand, have been developed by extending the documentation generation functionality that exports the tables to sheets of Microsoft Excel documents [P8].

5.4 Discussion

Hazards and their associated risks form the starting point of development of safety systems. In order to improve the support of UML AP for safety system development,

the modeling concepts were extended to enable the detailed presentations of hazards, to include safety related information in requirements and with support for documenting the usage of standard solutions.

In the hazard and risk modeling approach, the occurrences of hazards can be described with the well-known and intuitive FTA notation. FTA models enable depicting the fault and condition sequences that lead to the hazards. With the logical combinations of conditions, FTA models support the development of corrective functions, which can often be developed to stop the condition sequences leading to the hazards. In the approach, risk information, including the probabilities and consequences of hazards, is included in the hazards as their attributes. The tracing of hazards to requirements and further on, on the other hand, is supported with specific traces. RiskTreatments are used between hazards and requirements and TraceTelations between requirements and implementing design elements. By nature, the traces are one-directional but possible to query in order to support both forward and backward traceability.

Traceability, correctness and completeness are in the approach supported with both choices related to modeling notations and automated functions. When applicable, decisions on notations, such as the use of logic diagrams, have been made so that they are intuitive and informative to developers and comply with recommendations of safety standards. Logic diagrams, for example, can be used to depict the logic of safety requirements and they are already familiar to control system developers. They are also recommended by IEC 61508 and claimed to support several properties of systematic integrity, e.g. “correctness with respect to the safety needs to be addressed by software”. By using already recommended notations within MDD, the use of MDD techniques could also be easier to accept in safety system development. MDD of safety applications could, in this way, be seen as an application of existing, appropriate techniques in a new (MDD) context.

By including required, safety related information in models, it is also possible to automate the generation of part of the required documentation of safety functions and performing various consistency checks. Producing documentation is also a necessity in the application domain. Without MDD support, the documentation for certification purposes and for authorities, for example, would have to be produced manually. This would, however, significantly reduce the potential to benefit from MDD.

The presented automated functions generate, for example, traceability matrices for hazards and their risks as well as for requirements. In addition to presenting the traceability information in an intuitive form, the matrices support completeness by

warning developers about possibly overlooked hazards and requirements. Such are hazards and risks that are not traced further in the models. Automated consistency checks can and have been developed to warn users about possible, incompatible safety integrity levels related to requirements and safety system patterns, for example.

The starting point in the approach to use design patterns for safety system development is that the uses of patterns represent the design decisions of developers. As such, they should be deliberately marked in models, instead of trying to detect pattern instances in models, for example. In this way, reliable pattern information could be also used as part of the documentation. The presented SafetyPattern concepts enable modeling techniques and solutions that safety standards recommend. SafetyPatterns can have recommendations for different integrity levels. SafetyPatterns can also be collected into SafetyCatalogues with which it is possible to model recommendation tables of standards, for example.

Based on the SafetyPattern concepts, automated functions have been developed to extend the documentation generation possibilities. The functions enable collecting traceability information on the use of standard solutions. At the same time, they support correctness by comparing patterns that are used in models, and their recommendations, to catalogues and to the SILs required from the Packages in which they are used. User guidance has also been envisioned to support development work in a constructive manner, so that the tool could recommend techniques and solutions to be used, based on safety integrity requirements and selections on catalogues.

6 Summary of the Included Publications

The thesis includes eight publications. This Chapter presents the summaries of the publications and defines the contributions of the author in the publications.

[P1]

The paper presents an overview of the AUKOTON development process and UML Automation Profile (UML AP) as well as motivates the development of the UML AP tool. The tool builds on Topcased UML/SysML toolkit and implements UML AP modeling concepts as metamodel extensions to the UML and SysML metamodel implementations that are used by Topcased. The metamodel extensions are defined and implemented with Eclipse Modeling Framework (EMF) and graphical support with Topcased generators and Java programming. The concrete syntaxes of the new diagrams resemble traditional diagrams used in the domain. The tool utilizes the plug-in architecture of the Eclipse platform and implements a plug-in interface for finding and using model transformations that are required in model-driven development.

The author is the main author of the paper and responsible for the tool development in general. The development approach presented in the paper has been developed as a collaborative effort. The profile (UML AP) version supported by the original version of the tool had been previously developed at TUT in ASE.

[P2]

The paper presents the results of the industrial assessment of the AUKOTON development process and tools. The assessment was organized as a one-day event for industrial partners of the (AUKOTON) research project. In the event, industrial professionals utilized the developed process, modeling concepts as well as tools in an application development project and were observed and interviewed by researchers to collect qualitative material. According to the results, the techniques and tools could be successfully used in automation application development. The development process and tools were seen to automate some tasks that are currently performed manually. Interlocks were identified as an area for further improvements.

The author is the main author of the paper and responsible for the tool development in general. The development process has been a collaborative effort, the contributions of the author being in functional and platform specific modeling phases. The second author of the paper is the main responsible one for the research methodology used in the assessment event. The third author of the paper is the main responsible one for the

description of the industrial development process. The arrangements of the assessment have been a collaborative effort.

[P3]

The paper presents a method for transforming UML AP control application models to ModelicaML form to enable their simulation. The approach is targeted to model-in-the-loop simulations using a single simulation engine and it aims to transform and append control application models to plant simulation models. This enables design-time, closed-loop simulations of controlled processes in order to obtain the benefits of simulations early in the development process. The developed model transformation is implemented with QVT model transformation languages and packaged to a plug-in that connects to the tool platform with the extension interface of it. The case study used in the paper evaluates the interlock and control behavior of a controlled crane system.

The author is the main author of the paper and responsible for planning the paper, implementing required software prototypes, preparing and performing the reported simulations as well as writing the paper.

[P4]

The paper extends the simulation approach [P3] to modeling and simulation of sequentially executed control activities – Automation Sequences. The modeling approach is compared with UML state machines and the simulation approach extended from Modelica simulation of state machines. The paper completes the support of the simulation approach for all four aspects of basic control systems: binary and feedback control, sequential control and interlocks. The case study presented in the paper is related to paper industry. As a simulated process, it is the largest that has been so far used to evaluate the simulation approach. Based on the case study, it is argued that the approach scales to non-trivial industrial applications.

The author is the main author of the paper and responsible for planning the paper, implementing required software prototypes, preparing and performing the reported simulations as well as writing the paper.

[P5]

The paper presents a conceptual comparison of possible simulation approaches in model-driven development of automation and control applications. Additionally, the paper summarizes the observations from three simulation experiments in which the developed simulation approach has been used. In the comparison, the paper takes into consideration the benefits, restrictions and numbers of required simulation engines

related to simulation approaches. The influence of the domain and the practice of utilizing existing control system platforms are discussed. Related to the simulation approach of the author, the paper compares perceived benefits to those anticipated based on literature. According to the results, the approach is applicable to both machinery and process industry applications. In addition to the general benefits of simulations, the approach has enabled prototyping, experimenting and comparing control and interlock solutions. Furthermore, by enabling simulating early in the development process the approach detects inconsistencies in requirements and design.

The author is the main author of the paper and responsible for planning the paper, setting up and performing the comparison of the simulation approaches as well as writing the paper.

[P6]

The paper proposes model-driven development techniques as a means to facilitate the development of safety-related applications. UML Automation Profile is extended to the modeling of risks and hazards as well as to presenting how hazardous situations can occur. The information content is completed with traces to support traceability between design and development artifacts. Modeling concepts are rationalized to facilitate the understanding of software developers over the problem area and thus correctness of design. Model checks are used to support completeness so that risks, hazards and requirements are not overlooked later in design. Correctness, completeness and traceability are discussed with respect to their definitions in IEC 61508 (functional safety) standard.

The author is the main author of the paper and responsible for planning the paper, implementing required software prototypes as well as writing the paper.

[P7]

The paper presents an approach to model design-patterns and mark design pattern instances in models. The approach utilizes metamodel extensions for the both purposes and enables design pattern definitions to be collected to model libraries. The paper proposes the use of design patterns for documentation purposes in MDD and argues why design patterns could be especially valuable in MDD. In the presented approach patterns can be both traditional programming language level patterns as well as more general solutions, mainly describing the roles of entities in the patterns. A model transformation is developed that utilizes patterns to produce traceability documentation from models.

The author is the main author of the paper and responsible for planning the paper, implementing required software prototypes as well as writing the paper.

[P8]

The paper extends the design pattern modeling approach for SafetyPatterns - patterns of safety systems. Metamodel extensions are developed for specifying recommended safety levels for patterns and for organizing safety patterns to safety pattern catalogues. Such catalogues can be developed to correspond to recommendations of safety standards. Automated tool support is developed for checking the compliance of patterns (that are used in models) to safety levels required from the modeled applications. The tool support also enables comparing the models of applications to safety catalogues in terms of safety patterns in order to reveal inconsistencies and to suggest patterns that could be used.

The author is the main author of the paper and responsible for planning the paper, implementing required software prototypes as well as writing the paper.

7 Conclusions

The thesis discusses the use of MDD in automation and control application development. To study, whether or not their development could benefit from MDD, the thesis focuses on research topics that are related to modeling and developing tool support for modeling the applications, ability to integrate simulations into MDD and ability to include safety-related information in models. This Chapter summarizes the thesis, the research questions and the limitations of the work, and outlines future work.

7.1 Thesis Summary

The AUKOTON model-driven development process, which is used as a basis to be extended in the work, is introduced. Requirements for UML AP implementation and MDD tool support are derived from practical needs and special characteristics of the domain, including the re-use of platform specific solutions. UML, for which UML AP is an extension, can be extended for special purposes with two extension mechanisms. UML includes a built-in stereotype mechanism but can be extended also by extending the metamodel of the language with the use of MOF, on metamodeling layer M2.

The MOF based approach is selected for UML AP implementation. It provides stability for graphical tooling development and does not restrict the additions of metaclasses and meta-associations. At the same time, it enables the use of standard model transformation languages to implement model transformations. Open source modeling tools on Eclipse platform, which can be extended in order to re-use their support for plain UML and SysML, provide natural support for the MOF based approach. The tool used as a basis in UML AP tool implementation is Topcased. The three types of transformations that are required by AUKOTON can be added to the tool in a flexible manner using the Eclipse extension point mechanism and controlled with the user interface of the tool.

The re-use of existing solutions is in the approach supported with respect to both platform specific and platform independent modeling. Design patterns can be defined with specific pattern modeling concepts and their instances marked in models. Automated functions provide support for instantiating and marking patterns in models, for visualizing patterns in models and diagrams and for generating traceability information and statistics on the use of patterns. Platform specific blocks can be modeled in platform specific profiles that consist of stereotypes, tagged values and template AFs with block specific interfaces. With this information available in models, using platform specific blocks in code generation is straight-forward.

In order to support early testing and validation as well as to facilitate decision making, simulations are applied. Transformation assisted use of model-in-the-loop simulations within a single simulation engine is justified to enable simulating early but without requiring unnecessary simulation expertise of developers. A model transformation is developed that creates ModelicaML simulation counterparts of control application models and integrates them into existing plant (simulation) models. The results of the integration, ModelicaML simulation models of closed-loop systems, can be simulated using e.g. OpenModelica (open source) tools.

The simulation approach supports both platform independent and platform specific functions and is capable of creating application specific simulation classes based on Logic and Automation Sequence diagram definitions of AFs. The approach supports the different aspects of basic control systems: binary and feedback control, sequential control and interlocks. Observations from applying the simulation approach are presented. The size and complexity of the controlled processes have varied up to 1400 equations for a closed-loop system and demonstrated the scalability of the approach to non-trivial processes with industrial size and complexity. The simulations have enabled prototyping interlock and control solutions, revealed shortcomings in requirements and designs and enabled studying hazardous situations. These observed benefits are similar to those reported in literary. It is possible that many general benefits of simulation could be obtained also in MDD context but earlier than with more traditional simulation approaches.

The modeling approach is extended to include information that would be required for safety applications. The modeling of hazards, how they can be realized and their associated risks are supported with fault tree modeling. Fault trees are analytic and intuitive for both basic control and safety system developers. Their use can also facilitate the development of corrective functions, which can be designed to stop the fault and condition sequences that lead to hazardous situations. Hazards can be also traced to requirements that have been specified to treat them.

UML AP requirement concepts as well as the pattern modeling concepts are extended with safety-related information. Requirements can be refined with information on which safety level they must be fulfilled and which techniques, e.g. electrical, electronic or programmable electronic safety systems to use. Design pattern modeling concepts include SafetyPatterns, which are design patterns of safety systems and can include recommendations for applications of different levels. SafetyPatterns can be collected into catalogues to model recommendation tables of safety standards, for example.

The safety-related extensions are utilized by functions that gather traceability information and documentation to spreadsheets and at the same time automate checks of consistency. Hazards and their risks are traced to requirements and further to design that fulfils the requirements. Requirements that are not traced further to implementing design elements are identified automatically to improve completeness. Correctness is supported by, for example, consistency checks between used safety patterns and safety levels required from the applications. When appropriate, modeling techniques have been selected so that they comply with safety standards.

7.2 Research Questions Revisited

The research questions RQ1-3 are in the thesis addressed and discussed in Chapters 3, 4 and 5, respectively. Following is a summary of answers to the research questions.

RQ1: How to develop support for domain-specific, UML based modeling in control application development? How to develop support for and gain benefit from applying design patterns in models? How to enable and gain benefit from re-using platform specific blocks in modeling?

While the stereotype mechanism of UML would enable light-weight modifications, the use of MOF to extend the language metamodel on metamodeling layer M2 has no restrictions when domain-specific concepts are extensions to those of UML. Models conforming to MOF based metamodels can be processed with standard (QVT) model transformations so that references from transformation definitions to metamodels can be checked at compile-time. The MOF based extension approach is also supported by many open source tools, for example on Eclipse platform, so that graphical support for the domain concepts can be developed on top of an existing tool.

When design patterns are not restricted to the contents of Classifiers only, as in the UML approach, they can be supported by specifying and implementing suitable pattern concepts with, for example, the MOF based extension approach. With specific concepts for specifying and instantiating patterns, it is possible to develop automated support for using and benefitting from patterns. For example, applying patterns can be facilitated by the tool, pattern instances can be visualized for e.g. teaching and documentation purposes, and the traceability of solutions supported by analyzing patterns instances.

Design patterns can improve the re-use of platform independent design. However, in the domain an even more important characteristic is the re-use of platform specific implementation blocks, e.g. type circuits. To benefit from existing blocks in MDD, it

would need to be possible to use the blocks in the models so that the blocks could be instantiated to applications, parameterized and connected by code generators. In the developed approach, this is realized with an approach that uses stereotypes and their tagged values for identifying and parameterizing blocks as well as template AFs and their ports for the specification of block interfaces. Automated functions of the tool enable completing the interfaces of AFs to correspond to those of the blocks, when applying platform specific stereotypes.

RQ2: How can model-in-the-loop simulations be integrated into MDD of automation and control applications with UML based modeling? What are the requirements and constraints for selecting the simulation approach to be followed? How can simulations with the selected approach benefit MDD?

The developed transformation assisted simulation approach enables the use of MiL simulations early and using a single simulation engine, thus relieving developers from connecting simulation models and engines. To take into account the special characteristic of the domain related to re-use, the approach enables the use of simulation class libraries, in addition to creating new simulation classes based on Control Structure, Automation Sequence and Logic diagrams. Simulations can be used during both platform independent and platform specific phases of AUKOTON. The simulation transformation has been implemented with QVT and connected with the tool with use of the Eclipse extension point mechanism. The approach supports all the four aspects of basic control systems: binary and feedback control, sequential control and interlocks.

The supremacy of any simulation approach over others cannot be claimed based on the presented material. However, as simulations are already, in general, supported by control system vendors, to gain benefit from integrating simulations into MDD one should apply the MiL approach so that simulations could be used before generating code. The transformation assisted and co-simulation approaches to MiL simulations have varying requirements and constraints related to the simulatability and transformability of models, numbers of required transformations, the connectability of simulations and management of simulation cases, for example. With different techniques for e.g. modeling, MDD approaches can justifiably utilize different simulation approaches.

MDD techniques can be used for automating information transfer and repetitive tasks. The techniques, however, may not reduce the need to test, validate and compare designs, which are tasks to which simulations have been applied. MiL simulations may not provide all the benefits of simulations. However, they can be used early for

simulation tasks that do not require control system hardware. The developed simulation approach has been used in the development of both machinery and process industry applications. It has enabled prototyping and comparing control and interlock solutions, searching for control tunings and detecting inconsistencies in requirements and design. The approach has also enabled studying exceptional situations in a safe manner.

RQ3: How can the safety of control applications be supported in MDD? How can risk and hazard information be integrated into modeling? How can traceability, correctness and completeness be supported in models? How can the use of design patterns support documenting the safety features of control applications?

The techniques and solutions that improve safety and related quality attributes in traditional safety system development can improve them also in MDD. The information content of models can be extended to take into account requirements of safety standards related to, for example, traceability. With the safety information available in models, MDD techniques can be used for gathering the information to documentation and for automating consistency checks to promote properties such as traceability, correctness and completeness. When appropriate, modeling methods to be integrated into MDD can be selected according to recommendations of safety standards.

The modeling of hazards and their associated risks, which is the starting point in safety system development, is supported in many published extensions to UML. The UML AP approach uses the well-known FTA notation. The FTA modeling concepts have been implemented with MOF as metamodel extensions and given graphical tool support based on Topcased tool. With respect to the data content of models, hazards are traceable to requirements that have been specified to treat them.

Design patterns can be used to document practices and solutions to recurring problems also in safety system development. With specific safety pattern modeling concepts, patterns can include recommendations for applications of different safety integrity levels. Patterns, which are applied to models, and their recommendations for different levels can be compared with safety levels that are required from the applications. By modeling recommended techniques and measures as SafetyPatterns and recommendation tables as SafetyCatalogues, it is possible to automate checking whether or not recommended patterns have been applied and if not, which patterns should be applied to comply to a standard. In this way, SafetyPatterns can be used to guide the development.

7.3 Limitations and Future Work

The methods and techniques that are proposed in the thesis have been implemented on a prototype level, mainly on the open source Eclipse platform. The results obtained with constructive research, i.e. the solutions, methods, and techniques, have been assessed mainly experimentally and in a qualitative manner, to answer the research questions. However, the improvements obtained with the incrementally developed prototypes suggest their suitability and potential for improvements also in production use.

Assessing the results quantitatively, to obtain measures of their applicability to industrial problems and ability to increase efficiency, for example, would require their implementations in commercial quality tools. Developers would have to work with the tools and methods in a number of projects in order for the assessment results not to become biased by the developers being more experienced with their current tools and practices[16]. On the other hand, assessing the industrial applicability of the techniques with surveys would require the designer of the surveys to foresee all the relevant factors and the answerers to foresee the difficulties and benefits of techniques that they are not yet familiar with [P2], [103]. In [P2], this problem was solved by familiarizing industry professionals with the techniques with an example modeling project and by using two complementary methods for collecting qualitative material: participatory observation and interviewing. Apart from the results of the AUKOTON project, most developed research artefacts have been assessed by the author only, in example case studies reported in the included publications.

An important task in the future work would thus be to acquire industrial feedback about the industrial adoption potential of the techniques and their ability to facilitate the work of developers. Because of the number of implemented complex tools and techniques, an assessment event similar to the one used for [P2] could, however, require several days from industry professionals in order to familiarize them with the techniques. Another alternative to gather feedback and directions for improvements could be to work with a problem of industrial origin and to demonstrate work phases for professionals in order to provide a basis for discussion and interviewing. This approach could benefit from improvements to the code generator and from additional platform specific libraries of

[16] It is even possible that in order to get non-biased results, the developers would have to be trained for the techniques during their early careers.

the tool environment, so that an industrial control system platform could be used as a target.

The code generation support that has been developed into the tool environment should be updated to exploit the most recent advances in the modeling and simulation capabilities and to include I/O mappings. For example, Logic diagrams and Automation Sequence Diagrams, which are supported by the simulation integration, should be supported also by code generation. Simulation-assisted MDD would benefit from the ability to select test scenarios in a smart and systematic manner, based on, for example, requirements or hazards of the controlled process. At the same time, MDD techniques should support traceability between the tests and related model elements.

In addition, in order to assess industrial applicability and potential of the techniques, code generation support should be developed for an industrial control system platform. The type circuits of the platform should be possible to use in both application code and simulation models. With these future enhancements, it appears that control application development could benefit more from the developed MDD approach.

Bibliography

[1] IEC, 61508 functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission (2010).

[2] Iivari, J. A Paradigmatic Analysis of Information Systems As a Design Science. Scandinavian Journal of Information Systems 19(2007)2, pp. 39-64.

[3] Crnkovic, G.D. Constructive research and info-computational knowledge generation. In: Magnani, L., Carnielli, W. & Pizzi, C. (ed.). Model-Based Reasoning in Science and Technology. 2010, Springer. pp. 359-380.

[4] Hevner, A.R. & March, S.T. The information systems research cycle. Computer 36(2003)11, pp. 111-113.

[5] Gregor, S. & Hevner, A.R. Introduction to the special issue on design science. Information Systems and e-Business Management 9(2011)1, pp. 1-9.

[6] OMG. Model Driven Architecture (MDA) Guide. Object Management Group (2003).

[7] Booch, G. UML in action. Communications of the ACM 42(1999)10, pp. 26-28.

[8] Selic, B. UML 2: a model-driven development tool. IBM Systems Journal 45(2006)3, pp. 607-620.

[9] OMG. OMG Meta Object Facility (MOF) Core Specification 2.4.2. Object Management Group (2014).

[10] OMG. OMG Unified Modeling LanguageTM (OMG UML), Superstructure 2.4.1. Object Management Group (2011).

[11] OMG. OMG Unified Modeling LanguageTM (OMG UML), Infrastructure 2.4.1. Object Management Group (2011).

[12] OMG. OMG Systems Modeling Language (OMG SysML™) Version 1.3. Object Management Group (2012).

[13] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.0. Object Management Group (2008).

[14] Czarnecki, K. & Helsen, S. Classification of model transformation approaches. Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, 2003, Citeseer. pp. 1-17.

[15] Vepsäläinen, T., Hästbacka, D. & Kuikka, S. A Model-Driven Tool Environment for Automation and Control Application Development - Transformation Assisted, Extendable Approach. Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering. 2009, Tampere University of Technology. Department of Software Systems. pp. 315-329.

[16] Carrasco, J.A. & Dormido, S. Analysis of the use of industrial control systems in simulators: State of the art and basic guidelines. ISA transactions 45(2006)2, pp. 295-312.

[17] Shokry, H. & Hinchey, M. Model-based verification of embedded software. Computer 42(2009)4, pp. 53-59.

[18] Mattsson, S.E., Elmqvist, H. & Otter, M. Physical system modeling with Modelica. Control Engineering Practice 6(1998)4, pp. 501-510.

[19] MODELICA ASSOCIATION. Modelica ® - A Unified Object-Oriented Language for Systems Modeling, Language Specification, Version 3.3. Modelica Association (2012).

[20] Schamai, W. Modelica Modeling Language (ModelicaML): A UML Profile for Modelica. 2009, Linköping University Electronic Press.

[21] Schamai, W., Fritzson, P., Paredis, C. & Pop, A. Towards unified system modeling and simulation with ModelicaML: modeling of executable behavior using graphical notations. Proceedings of the 7th International Modelica Conference, 2009, pp. 612-621.

[22] Herrmann, D.S. Software safety and reliability. Los Alamitos, California 1999, IEEE Computer Society.

[23] IEC. 62061 Safety of machinery: Functional safety of electrical, electronic and programmable electronic control systems. International Electrotechnical Commission (2005).

[24] IEC. 61513 Nuclear power plants - Instrumentation and control important to safety - General requirements for systems. International Electrotechnical Commission (2011).

[25] Leveson, N.G. System safety in computer-controlled automotive systems. SAE transactions 109(2000)7, pp. 287-294.

[26] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design patterns: elements of reusable object-oriented software. 1994, Addison-Wesley.

[27] Douglass, B.P. Real-time design patterns: robust scalable architecture for real-time systems. 2003, Addison-Wesley.

[28] Hanmer, R. Patterns for fault tolerant software. 2013, John Wiley & Sons.

[29] Saridakis, T. Design patterns for checkpoint-based rollback recovery. Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP), 2003.

[30] IEC. 61131-3: Programmable Controllers - Part 3, Programming Languages. International Electrotechnical Commission (2013).

[31] IEC. 61499-1: Function Blocks-Part 1: Architecture. International Electrotechnical Commission (2012).

[32] Thramboulidis, K.C. & Tranoris, C.S. Developing a CASE tool for distributed control applications. The International Journal of Advanced Manufacturing Technology 24(2004)1-2, pp. 24-31.

[33] Tranoris, C. & Thramboulidis, K. A tool supported engineering process for developing control applications. Computers in Industry 57(2006)5, pp. 462-472.

[34] Strasser, T., Ebenhofer, G., Rooker, M. & Hegny, I. Domain-specific design of industrial automation and control systems: The MEDEIA approach. Intelligent Manufacturing Systems, 2010, pp. 18-23.

[35] Vyatkin, V., Hanisch, H., Pang, C. & Yang, C. Closed-loop modeling in future automation system engineering and validation. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 39(2009)1, pp. 17-28.

[36] Dubinin, V., Vyatkin, V. & Pfeiffer, T. Engineering of validatable automation systems based on an extension of UML combined with function blocks of IEC 61499. IEEE International Conference on Robotics and Automation, 2005, pp. 3996-4001.

[37] Hussain, T. & Frey, G. Defining IEC 61499 compliance profiles using UML and OCL. 5th IEEE International Conference on Industrial Informatics, 2007, pp. 1157-1162.

[38] Panjaitan, S. & Frey, G. Development process for distributed automation systems combining UML and IEC 61499. International Journal of Manufacturing Research 2(2007)1, pp. 1-20.

[39] Thompson, H., Ramos-Hernandez, D., Fu, J., Jiang, L., Choi, I., Cartledge, K., Fortune, J. & Brown, A. A flexible environment for rapid prototyping and analysis of distributed real-time safety-critical systems. Control Engineering Practice 15(2007)1, pp. 77-94.

[40] Marcos, M., Estévez, E., Gangoiti, U., Sarachaga, I. & Barandiarán, J. UML modelling of industrial distributed control systems. Proceedings of the Sixth Portuguese Conference on Automatic Control, 2004, pp. 127-132.

[41] Lukman, T., Godena, G., Gray, J., Heriŏko, M. & Strmŏnik, S. Model-driven engineering of process control software–beyond device-centric abstractions. Control Engineering Practice 21(2013)8, pp. 1078-1096.

[42] Witsch, D. & Vogel-Heuser, B. Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions. IEEE Conference on Emerging Technologies & Factory Automation, 2009, pp. 1-6.

[43] Vogel-Heuser, B., Witsch, D. & Katzke, U. Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. International Conference on Control and Automation, 2005, IEEE. pp. 1034-1039.

[44] France, R.B., Kim, D., Ghosh, S. & Song, E. A UML-based pattern specification technique. IEEE Transactions on Software Engineering 30(2004)3, pp. 193-206.

[45] France, R., Chosh, S., Song, E. & Kim, D. A metamodeling approach to pattern-based model refactoring. IEEE Software 20(2003)5, pp. 52-58.

[46] Kajsa, P. & Majtás, L. Design patterns instantiation based on semantics and model transformations. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J. & Rumpe, B. (ed.). SOFSEM 2010: Theory and Practice of Computer Science. 2010, Springer. pp. 540-551.

[47] Tsantalis, N., Chatzigeorgiou, A., Stephanides, G. & Halkidis, S.T. Design pattern detection using similarity scoring. Software Engineering, IEEE Transactions on 32(2006)11, pp. 896-909.

[48] Briand, L.C., Labiche, Y. & Sauve, A. Guiding the application of design patterns based on uml models. 22nd IEEE International Conference on Software Maintenance, 2006, pp. 234-243.

[49] Dong, J. UML extensions for design pattern compositions. Journal of object technology 1(2002)5, pp. 151-163.

[50] Jing, D., Sheng, Y. & Kang, Z. Visualizing design patterns in their applications and compositions. IEEE Transactions on Software Engineering 33(2007)7, pp. 433-453.

[51] Hästbacka, D., Vepsäläinen, T. & Kuikka, S. Model-driven development of industrial process control applications. Journal of Systems and Software 84(2011)7, pp. 1100-1113.

[52] Ritala, T. & Kuikka, S. UML automation profile: enhancing the efficiency of software development in the automation industry. 5th IEEE International Conference on Industrial Informatics, 2007, pp. 885-890.

[53] IEC. 62424: Representation of Process Control Engineering – Requests in P&I Diagrams and Data Exchange Between P&ID Tools and PCE-CAE Tools. International Electrotechnical Commission (2008).

[54] OMG. UML™ Profile for Schedulability, Performance, and Time Specification, Version 1.1. Object Management Group (2005).

[55] OMG. UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification, Version 1.1. Object Management Group (2008).

[56] OMG. Object Constraint Language, Version 2.4. Object Management Group (2014).

[57] Hästbacka, D. Developing Modern lndustrial Control Applications: On lnformation Models, Methods and Processes for Distributed Engineering. Doctoral Thesis. Tampere University of Technology. Publication 1143 (2013).

[58] Alexander, C., Ishikawa, S. & Silverstein, M. A Pattern Language: Towns, Buildings, Construction. 1977, Oxford University Press.

[59] Alexander, C. The timeless way of building. 1979, Oxford University Press.

[60] Agerbo, E. & Cornils, A. How to preserve the benefits of design patterns. ACM SIGPLAN Notices, 1998, ACM. pp. 134-143.

[61] Karaila, M. & Systa, T. Applying Template Meta-Programming Techniques for a Domain-Specific Visual Language - An Industrial Experience Report. 29th International Conference on Software Engineering, 2007, IEEE. pp. 571-580.

[62] Evesti, A. Quality-oriented software architecture development. VTT publications 636(2007).

[63] Vepsäläinen, T. UML-Profiilityökalu Automaatiosuunnitteluun. M.Sc. Thesis. Tampere University of Technology (2008).

[64] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. Pattern oriented software architecture: a system of patterns. 1996, John Wiley & Sons.

[65] Appleton, B. Patterns and software: Essential concepts and terminology. Object Magazine Online 3(1997)5, pp. 20-25.

[66] MODELISAR Consortium Functional Mock-up Interface for Co-simulation, Document version 1.0. MODELISAR (2010).

[67] Hemingway, G., Neema, H., Nine, H., Sztipanovits, J. & Karsai, G. Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach. Simulation 88(2012)2, pp. 217-232.

[68] Hegny, I., Wenger, M. & Zoitl, A. IEC 61499 based simulation framework for model-driven production systems development. 15th IEEE Conference on Emerging Technologies and Factory Automation, 2010, IEEE. pp. 1-8.

[69] Yang, C. & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. Control Engineering Practice 20(2012)12, pp. 1259-1269.

[70] Vyatkin, V., Karras, S. & Pfeiffer, T. Architecture for automation system development based on IEC 61499 standard. 3rd IEEE International Conference on Industrial Informatics, 2005, IEEE. pp. 13-18.

[71] Ferrarini, L. & Dedè, A. A model-based approach for mixed hardware in the loop simulation of manufacturing systems. 10th IFAC Workshop on Intelligent Manufacturing Systems, 2010, IFAC. pp. 36-41.

[72] Thompson, H., Ramos-Hernandez, D., Fu, J., Jiang, L., Nu, J. & Dobinson, D. The FLEXICON co-simulation tools applied to a marine application. Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment 222(2008)2, pp. 81-94.

[73] Herzner, W., Schlick, R., Schlager, M., Leiner, B., Huber, B., Balogh, A., Csertan, G., LeGuennec, A., LeSergent, T. & Suri, N. Model-based development of distributed embedded real-time systems with the decos tool-chain. SAE AeroTech Congress & Exhibition, 2007.

[74] Plummer, A. Model-in-the-loop testing. Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering 220(2006)3, pp. 183-199.

[75] Chae, H., Jin, X., Lee, S. & Cho, J. TEST: Testing Environment for Embedded Systems Based on TTCN-3 in SILS. In: Ślęzak, D. et al. (ed.). Advances in Software Engineering. 2009, Springer. pp. 204-212.

[76] Short, M. & Pont, M.J. Assessment of high-integrity embedded automotive control systems using hardware in the loop simulation. Journal of Systems and Software 81(2008)7, pp. 1163-1183.

[77] Stoeppler, G., Menzel, T. & Douglas, S. Hardware-in-the-loop simulation of machine tools and manufacturing systems. Computing and Control Engineering 16(2005)1, pp. 10-15.

[78] Dougall, D.J. Applications and benefits of real-time IO simulation for PLC and PC control systems. ISA transactions 36(1997)4, pp. 305-311.

[79] Karhela, T. A software architecture for configuration and usage of process simulation models: Software component technology and XML-based approach. Doctoral Thesis. VTT Technical Research Centre of Finland. VTT Publications 479 (2002).

[80] Fritzson, P. & Bunus, P. Modelica - a General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation. 35th Annual Simulation Symposium, 2002, IEEE. pp. 365-380.

[81] Schamai, W., Fritzson, P. & Paredis, C.J. Translation of UML state machines to Modelica: Handling semantic issues. SIMULATION: Transactions of The Society for Modeling and Simulation International 89(2013)4, pp. 498-512.

[82] Schamai, W., Pohlmann, U., Fritzson, P., Paredis, C.J., Helle, P. & Strobel, C. Execution of UMLState Machines Using Modelica. 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (In Conjunction with MODELS), 2010, Citeseer. pp. 1-10.

[83] Vepsäläinen, T. & Kuikka, S. Simulation-Based Development of Safety Related Interlocks. In: Pina, N., Kacprzyk, J. & Filipe, J. (ed.). Simulation and Modeling Methodologies, Technologies and Applications. 2013, Springer. pp. 165-182.

[84] Vepsäläinen, T., Hästbacka, D. & Kuikka, S. Simulation Assisted Model-Based Control Development - Unifying UML AP and Modelica ML. 11th International Middle Eastern Simulation Multi Conference, 2010, EUROSIS.

[85] Huber, B. & Obermaisser, R. Model-based development of integrated computer systems: Modeling the execution platform. 5th Workshop on Intelligent Solutions in Embedded Systems, 2007, IEEE. pp. 151-164.

[86] Biehl, M., DeJiu, C. & Törngren, M. Integrating safety analysis into the model-based development toolchain of automotive embedded systems. ACM Sigplan Notices 45(2010)4, pp. 125-132.

[87] Guillerm, R., Demmou, H. & Sadou, N. Information model for model driven safety requirements management of complex systems. In: Aiguier, M., Bretaudeau, F. & Krob, D. (ed.). Complex Systems Design & Management. 2010, Springer. pp. 99-111.

[88] Zoughbi, G., Briand, L. & Labiche, Y. A UML profile for developing airworthiness-compliant (RTCA DO-178B), safety-critical software. In: Engels, G., Opdyke, B., Schmidt, D.C. & Weil, F. (ed.). Model Driven Engineering Languages and Systems. 2007, Springer. pp. 574-588.

[89] Zoughbi, G., Briand, L. & Labiche, Y. Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile. Software & Systems Modeling 10(2011)3, pp. 337-367.

[90] Douglass, B. Analyze system safety using UML within Telelogic Rhapsody environment. 2009, White Paper, Rational Software, IBM Software Group.

[91] IEEE. 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. IEEE (2000).

[92] Douglass, B.P. Safety-critical systems design. Electronic engineering 70(1998)862, pp. 45-50.

[93] Rauhamäki, J., Vepsäläinen, T. & Kuikka, S. Functional Safety System Patterns. VikingPLoP 2012 Conference, 2012, pp. 48-68.

[94] Rauhamäki, J. & Kuikka, S. Patterns for Control System Safety. 18th European Conference on Pattern Languages of Programs, 2013, ACM.

[95] Kruchten, P., Lago, P., van Vliet, H. & Wolf, T. Building up and exploiting architectural knowledge. 5th Working IEEE/IFIP Conference on Software Architecture, 2005, IEEE. pp. 291-292.

[96] Sklet, S. Safety barriers: Definition, classification, and performance. Journal of Loss Prevention in the Process Industries 19(2006)5, pp. 494-506.

[97] Möller, N. & Hansson, S.O. Principles of engineering safety: risk and uncertainty reduction. Reliability Engineering & System Safety 93(2008)6, pp. 798-805.

[98] de Miguel, M.A., Briones, J.F., Silva, J.P. & Alonso, A. Integration of safety analysis in model-driven software development. IET Software 2(2008)3, pp. 260-280.

[99] IEC. 61804-1: Function blocks (FB) for process control - Part 1: Overview of system aspects. International Electrotechnical Commission (2006).

[100] Peltola, J., Sierla, S., Vepsalainen, T. & Koskinen, K. Challenges in industrial adoption of model-driven technologies in process control application design. Industrial Informatics (INDIN), 2011 9th IEEE International Conference on, 2011, pp. 565-572.

[101] ISO. 13849-1:2006 Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design. International Organization for Standardization (2006).

[102] Vepsäläinen, T., Kuikka, S. & Eloranta, V. Software architecture knowledge management for safety systems. 17th IEEE International Conference on Emerging Technologies & Factory Automation, 2012, IEEE. pp. 1-8.

[103] Trauth, E.M. The choice of qualitative methods in IS research. In: Trauth, E. (ed.). Qualitative research in IS. 2001, IGI Publishing. pp. 1-19.

Publications

Publication 1

Vepsäläinen, T., Hästbacka, D., Kuikka, S. (2008) Tool Support for the UML Automation Profile – for Domain-Specific Software Development in Manufacturing. Proceedings of the 3rd International Conference on Software Engineering. Sliema, Malta, October 26-31, 2008, pp 43-50. IEEE Computer Society, 2008.

DOI: 10.1109/ICSEA.2008.22

Tool Support for the UML Automation Profile - for Domain-Specific Software Development in Manufacturing

Timo Vepsäläinen, David Hästbacka, Seppo Kuikka
Tampere University of Technology, Department of Automation Science and Engineering,
P.O. Box 692, FIN-33101 Tampere, Finland
{timo.vepsalainen, david.hastbacka, seppo.kuikka}@tut.fi

Abstract

The development of modern distributed automation applications is challenging and present development practices contain manual transferring of informal information from one phase to another. Our research aims to overcome some of these challenges by integrating concepts from modern object-oriented design, model-driven development and high-level modeling potential of the UML automation profile into a seamless development path from PI-diagrams to control software. This paper presents a prototype of a control engineering tool that supports the UML automation profile and is intended to cover part of the development chain. The tool was implemented on the Eclipse platform and it utilizes various open source tools and frameworks to enable also usage of UML and SysML in modeling work. The implemented tool can be extended by transformation tools capable of processing requirements of the control system and PIM-model of the designed control software.
Keywords: automation, control, MDE, UML profile, modeling tool.

1. Introduction

Automation is important in manufacturing for various reasons, such as achieving or enhancing the quality and flexibility of industrial production and the safety and efficient use of manufacturing equipment. Strict requirements concerning dependability, safety and various other quality issues are consequently laid on the systems. The fundamental role of automation applications is in the measurement and control of a system or process, usually leaving for humans only the overall supervision of the integrated and distributed automation system.

Software has an essential role in developing modern industrial automation applications. In the recent years, the challenges in software development for automation systems have increased, because the systems have become more and more complex while competition within the automation industry has tightened. In order to maintain competitiveness, automation system and application vendors have been forced to increase both the efficiency and openness of the development process.

A potential solution is the use of high-level domain-specific modeling languages and tools that are capable of capturing the essential concepts of modern, complex automation applications. Key issues for such languages are the capability of the modeling language to address the safety- and mission-critical properties of automation systems. Preferably, the language shall be such that it can exploit the appropriate concepts and notations of generic information system modeling, i.e. UML and relevant profiles. As a result, a high-level modeling language, based on UML V2 and suitable UML profiles, provides substantial potential for modeling automation applications in an efficient and vendor independent way.

The tools shall give support for the entire application development life cycle – starting from manufacturing oriented requirements and proceeding via platform independent architectural considerations of an automation application to the platform specific and device oriented implementation.

This paper will first consider domain-specific languages as well as UML profiles and metamodels in section 2, Related work. It will then discuss in section 3, the concept of a seamless development process of automation applications, which we in TUT (Tampere University of Technology) are developing within the so called AUKOTON project, in collaboration with automation companies, VTT (Technical Research Centre of Finland) and HUT (Helsinki University of

Technology) in Finland. Section 4 gives an overall view of the automation profile and section 5 of the Automation Profile Tool (AP-Tool), both developed in TUT / ASE (Department of Automation Science and Engineering). Section 6 gives more details of the present AP-Tool. Section 7 concludes the research work and discusses future work to enhance the profile, the AUKOTON chain and the tool.

2. Related work

Domain-specific languages and DSL development methods raise the level of *abstraction* in software development and problem solving by leveraging *domain-specific* concepts and practices as a modeling or programming language. Contrary to general-purpose programming languages, such as C and Java, domain-specific programming languages are dedicated to a particular problem domain, thus making it easier to solve problems while enhancing properties such as quality and reliability among others. As a drawback the design and implementation of domain-specific programming languages is challenging and requires a lot of effort and consideration.

However, by using a domain-specific language, problems can be solved on a higher abstraction level with concepts and practices of the specific domain, that is to say in a more problem-oriented manner. This does not only increase productivity but also makes reuse of solutions more feasible and most importantly enables domain experts to work with familiar building blocks and notions.

In the area of automation and control applications as well as many other specialized domains there are recurring concepts and structures that encourage the use of domain-specific modeling and programming languages. In contrast to more general modeling (e.g. UML) and traditional software development, the DSL methods not only bring the elements of a particular domain but they may also restrict development to a very narrow realm in that domain. A common trend with systems in general is increased complexity and a strong need for integration on all levels. This calls for modeling various objects and systems on different levels of abstraction and the ability to easily extend modeling and development to new concepts and related systems. Besides the domain-specific concepts in the multidisciplinary area of automation and control applications there is also a need for extendibility and more general modeling and development capabilities.

In recent years, there has been wide industrial and academic interest in UML profiles and their usage in development and design work. An example of industrial interests is the profile produced by the AUTOSAR project. The UML profile for AUTOSAR is intended to provide a precise and pragmatic mapping between concepts of the UML 2.0 and the AUTOSAR metamodel [1].

An example of usage of UML in the automation domain is the work carried out by Thramboulidis and Tranoris who have integrated UML and the function block concept of the automation domain by introducing a set of transformation rules to generate function block applications from UML models [4]. Thramboulidis has also introduced a hybrid approach combining UML and the function block concept for distributed control system development. The study also examined applicability of the standardized *UML profile for Schedulability, Performance and Time* to the proposed development process [5]. Compared to our work, the studies carried out by Thramboulidis et al. rely more on general modeling capabilities of UML, instead of common modeling elements and terminology of the automation domain.

3. AUKOTON development chain

The AUKOTON project introduces a *development chain* for industrial software development, the AUKOTON chain, which utilizes advanced practices with formal requirement description, profile assisted UML modeling and automatic code generation.

Our observations during interviews and previous studies with automation and control engineering professionals indicate that current practices contain stages in which informally represented information is transferred from one phase to another. These stages contain a lot of manual work and are therefore prone to error while requirements and additional information needed to satisfy the required functionality is supplemented. Another important observation is companywise varying development processes and the lack of common application modeling concepts. As a result interoperability suffers between development processes of different companies, the design process becomes platform dependent early in the development, and the reuse of solutions is difficult.

The seamless AUKOTON development chain combines foremost industrial practices with object-oriented design and concepts from *model-driven development* (MDD). AUKOTON aims to conceptualize a structured and formal representation of initial requirements and source information as well as design requirements, architectural descriptions, and information concerning implementation details to automatically build a running application. Compared to traditional application development practices the AUKOTON chain highlights the importance of

platform independent modeling, automatic transfer of design information, such as process instrumentation lists, and late binding of platform specific implementation details, in order to enhance productivity, solution reuse and software quality.

MDD and Model Driven Engineering (MDE) put emphasis on the use of models as primary engineering artifacts in the development process. These models are then refined from one stage to another either in a user assisted manner or automatically. The background data traditionally guides decisions made during each phase but it may also contain information that can be automatically processed and transferred to the next design phase. For automation and control applications this could be for instance connections between different parts of the system or selecting a particular low-level function block for the target DCS (Distributed Control System) based on information in initial P&I diagrams.

Model transformations are an essential part of the MDE concept and usually related to metamodeling methods. Model transformations take source models as input and produce target models as a result. These models typically conform to given metamodels that can be understood as higher level models that define the semantics and the syntax of a modeling language.

The AUKOTON chain consists of a starting data phase where initial requirements and background information from sources such as P&I diagrams, process descriptions, and instrumentation lists are gathered into a formal structured representation. In the following phase where most of the actual design work is done the UML automation profile (UML AP) is utilized (see section 4). The UML modeling results in a *platform independent model* (PIM) that is used as a source model in the transformation in the final stage. The PIM can be tagged with stereotypes to indicate additional information needed for the transformation that produces a *platform specific model* (PSM) of the application. The PSM is a representation of the application based on building blocks and constructs of the target control platform and is used to generate the final executable application. The AUKOTON chain described is illustrated in figure 1.

In order to support the development process presented a tool based on the Eclipse framework has been developed. At this moment the tool implements the UML automation profile concepts and supports application modeling covering requirements, application functionality, devices and resources as well as application architecture and distribution. In addition, the tool offers well-defined extension points for integration to plug-ins that fulfill the rest of the AUKOTON chain (see figure 1). By using this type of a loosely coupled tool chain, responsibilities can be shared conveniently and the approach can easily be adapted to, for instance, new *target platforms*.

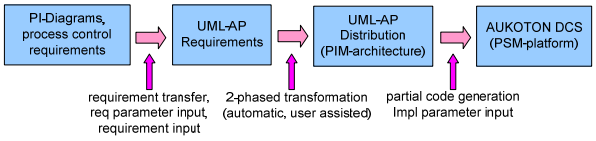

Fig. 1. The AUKOTON chain utilizes current practices with formal requirement description, UML modeling and automatic code generation.

4. The UML automation profile

The main principles of the UML automation profile, which is specified in detail in [3], are presented in this section as background information for the tool presentation in sections 5 and 6. The utilization of the relevant existing UML profiles is also indicated. The automation profile is divided into several areas of responsibility, each covering an important automation domain specific aspect. Such a division is made in order to allow easier management and further enhancement of the profile. In addition, the designers using the profile will be able to easily adopt and utilize only those parts of the profile, found relevant for the needs of the application.

The profile is organized into *four independent subprofiles*, as depicted in Fig. 2. Case problems that represent typical (but still as diversified as possible) automation applications have been utilized as test cases for the profile.

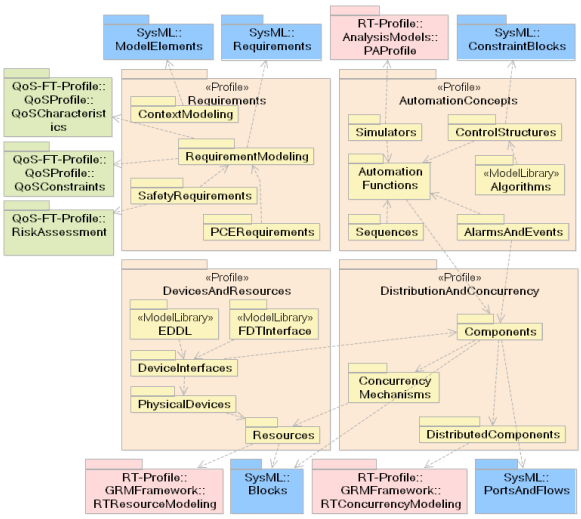

Fig. 2. The high-level organization of the UML automation profile.

The profile consists of stereotypes, tagged values and constraints, which are the principal means for introducing domain-specific concepts and their semantics into the language. The stereotypes are extended, as extensively as possible, from feasible elements of the UML Real-Time Profile, SysML and UML Profile for Quality of Service and Fault Tolerance.

The *Requirements subprofile* provides means for organizing the original requirements in a graphical fashion. The graphical representation is lightweight by emphasizing the relations between requirements and giving only a summary for each requirement. This summary allows identifying, rationalizing, categorizing and shortly describing the requirement. The requirement specification diagram specializes the requirement diagram of SysML. The purpose of the diagram type is to enable specifying automation application's requirements and linking requirements to actual model structures and design decisions.

The *AutomationConcepts subprofile* provides support for specifying various kinds of functionalities and capabilities common in the applications of the automation domain. One of the key concepts of the subprofile is the automation function that includes a category of common domain-specific functionalities. The functionalities covered in this subprofile include control of devices, sequential control, handling of events and alarms, as well as executing simulations. In addition, the subprofile supports the specification of control algorithms and integrating them with control functionalities.

The AutomationConcepts subprofile approaches control functions with the so-called *control structure diagram,* based on the SysML block definition diagram*,* that illustrates a control structure, or a set of structures, by linking the control outputs, inputs and algorithms together.

Automation applications often perform sequential control activities, such as start-up or shutdown sequences or batch automation sequences, which consist of activities initiated by measured state or timing conditions. Sequential control consists of a collection of sequential steps (or procedural elements) using which, a specific control task is executed. Indeed, this subprofile defines an *automation sequence diagram*, based on UML state machines, that allows the building of sequences from state-like steps, each with dedicated tasks executed when the step is entered, exited or during the step.

The *DistributionAndConcurrency subprofile* supports the specification of so-called automation components, from which an automation application software is composed, and the deployment of these components to active resources, such as controllers. Each resource may execute several components, thus supporting the distribution models of industrial standards. In order to allow communication between the components over the network in a transparent manner, the profile allows the utilization of several common communication patterns, including the publisher-subscriber and client-server patterns.

Another issue addressed in this subprofile is concurrency that causes the need for mutual exclusion and synchronization. The subprofile allows protection of resources with mutual exclusion mechanisms. Synchronization mechanisms may also be specified. For covering general concurrency issues, the package depends on the concurrency subprofile of the UML RT-profile.

The *DevicesAndResources subprofile* provides support for interfacing with various kinds of automation devices by specifying device interfaces and I/O operations in a platform and hardware independent manner. A device interface includes only the necessary information for identifying and interacting with the device, such as an identifier and type of the hardware interface. The subprofile also takes into account that interaction with a device may take place with any number of signals. Therefore, a device interface contains a set of ports, each transferring a single signal.

5. UML AP Tool design approach

The design and implementation work of a tool supporting the UML automation profile, the UML AP Tool, was initiated and influenced mainly by two factors. Firstly, the UML automation profile was designed to be used by *automation and control engineers* to design and to specify automation applications. Tool support was needed to further improve and to experimentally estimate the automation profile. Secondly, the AUKOTON chain requires tool support for the automation profile as two stages of the modeling chain use concepts of the profile. Moreover, the AUKOTON chain requires support for other functionalities regarding the model driven characteristics of the tool chain.

Besides defining a high-level modeling language for software-oriented applications in the automation domain, the automation profile also makes demands to tools supporting it. In addition to the profiles own concepts, practical usage of the profile requires tools to enable the use of any diagram type or modeling concept of UML and SysML [3]. Even though the profile defines three new diagram types, not all of the concepts of it are intended to be used in the new

diagram types only. For example, most of the concepts defined in the DevicesAndResources sub-profile fit most naturally as stereotypes to specialize semantics of the modeling concepts of UML and SysML. These concepts do have specialized semantics, for example regarding to roles in certain *design patterns*, and usually sets of attributes related to the semantics but their structural characteristics are not rich enough to model all the desired aspects of the application.

The automation profile defines most of the concepts needed in its diagram types but does not actually specify the concrete syntax of the diagram types and graphical presentation of the elements. The intended users of the tool, however, are automation and control engineers accustomed to traditional diagram types of the automation domain. Because of this, the diagram types were chosen to be implemented to resemble these traditional diagrams with the intention to help intended users to familiarize themselves with the tool.

There are several structural similarities between the three own diagram types of the automation profile and certain diagram types of UML and SysML. Despite the similarities, some of the concepts needed in the aforementioned new diagram types have structural features that do not fit to the UML metamodel. That is to say, implementing these concepts requires at least new meta-associations, in addition to defining stereotypes, to be added to the metamodels of UML and SysML.

The light weight profile mechanism of UML is based on stereotypes that can be used to specialize semantics of the modeling language concepts. Stereotypes may also define tagged values (attributes) that can be used to parameterize semantic characteristics of the stereotypes. Stereotypes cannot, though, be used in a way that would contradict with the semantics of the UML metamodel [2]. The UML super structure specification, for example, denies stereotypes to be used to insert new meta-associations between metaclasses and adding new metaclasses, which is exactly what the precise implementation of the automation profile would require. Fortunately, there are no such restrictions imposed to possible modifications to be done on the MOF-based (Meta Object Facility) M2 metamodel layer on which the abstract syntax of UML is defined.

These aspects imply that in the automation profile implementation some of its concepts require modifications on the M2 metamodel layer while others are more naturally supported with stereotypes which are modeling-time constructs. This, of course, has influence on implementation techniques. Additionally, the implementation, including metamodel implementation, needed to be tightly integrated to an existing UML/SysML tool as it would require an excessive amount of development work to implement also UML and SysML support from scratch.

Mostly because of these needs to integrate the tool to an existing tool and to modify the metamodel being used, the tool was chosen to be implemented on the *Eclipse platform*. On the Eclipse platform, there exists a variety of open source UML/SysML tools which also use the same open source implementation of the UML metamodel, org.eclipse.uml2. The UML metamodel implementation is itself based on the open source *Eclipse Modeling Framework* (EMF) that can also be used to make further additions to the metamodel. Eclipse and EMF are also a good choice considering the AUKOTON chain and its need for M2M (model to model) transformations. There are several transformation tools supporting EMF, such as SmartQVT and Atlas Transformation Language (ATL).

The AUKOTON tool chain, as presented in section 3 and Fig. 1, processes the model of the application to be implemented and its requirements on four layers of abstraction. Two of these four representations of the system use concepts defined by the automation profile and needs the support of the UML AP Tool. In addition, the tool needs to support *importing* requirements from design phases and tools preceding software design and *exporting* models to code generators.

By using so called extension points, import and export functionalities can be integrated to the tool and activated from the user interface of the tool which is more convenient than using entirely independent tools. The same user interface can also be used to select the parts of the model that will be processed in the transformation.

6. UML AP Tool development

The UML AP Tool is a result of design and implementation work based on features and requirements presented in the previous section. The present version of the tool was finished during spring 2008 but the tool will be further developed during the AUKOTON project because of the needed enhancements to the profile and additional features needed to support the AUKOTON chain.

The tool was implemented on the Eclipse platform to extend the open source Topcased UML/SysML tool (one of the few tools supporting UML, SysML and further diagram development at the time the development work was started). The Topcased tool also supports implementing own diagram types on EMF-based metamodels. The support includes

generator models for generating skeletons of graphical editors and diagram type implementations which can be easily modified and integrated to the Topcased tool environment.

As presented in section 5, the implementation of the automation profile required also modifications to the implementation of the metamodel used by the tool. In this case these modifications were restricted to additions of new metaclasses and meta-associations corresponding to concepts defined by the automation profile. Additions could be realized in a distinct plug-in that is only dependent of the existing implementations of UML and SysML metamodels, the org.eclipse.uml2 and the org.topcased.sysml.

Both of the extended metamodel implementations are based on the Eclipse Modeling Framework (EMF) which was also used to implement the additional parts of the metamodel. The additions, new meta-associations and metaclasses extending metaclasses of UML and SysML, were specified in an ecore model (MOF-like metamodel in EMF) which was further used to generate implementing Java classes and Eclipse plug-ins. Concepts not realized with metamodel extensions are supported with light weight UML profile consisting of stereotypes corresponding to the concepts of the profile. In the present version of the UML AP Tool, the modeling-time profile is also imported automatically to all new models in order to ease usage of the stereotypes.

The UML AP Tool consists of seven Eclipses *plug-ins*, each implementing part of the functionality of the whole application. The plug-ins follow Topcased's naming convention and they are presented in Fig 3 that also presents the most important (but not all) couplings of the plug-ins to the extended Topcased tool and to the Eclipse platform.

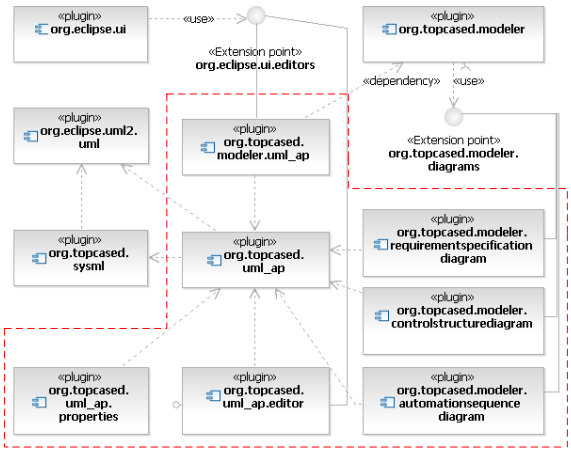

Fig. 3. The seven plug-ins of the UML AP Tool and the most important couplings to the extended Topcased tool and the platform.

The central plug-in in the picture, and in the context of the tool, is the plug-in implementing the additions to the metamodel, org.topcased.uml_ap. Plug-in org.topcased.uml_ap.editor is a simple tree editor capable of handling models conforming to the metamodel.

The org.topcased.modeler plug-in is the graphical editor plug-in of the tool. In addition to basic graphical editors functionality, the modeler plug-in also defines its own *extension points* of the tool to enable external transformation plug-ins usage from the tool. The modeler plug-in is based on the graphical editor generation of Topcased's and is basically a GEF (Graphical Editing Framework) editor.

The own *extension interface* of the tool, defined by the modeler plug-in, consists of import and export extension points. Performing the import functionality can be initiated by the user of the tool and the user may also select the desired RequirementDefinition from tools Outline view and the desired import implementation from the list provided by the tool. The export extension point is technically very similar to the import extension point and allows exporting Packages (UML::Package) or elements of any type extending the Package type.

Both the import and the export extension points allow mediating references to elements of the processed model and initiating the import or export functionality, respectively. The mediated parts of the model, however, are not separated from the whole model. Because of this, the choice to use the referenced element and children of it or the whole model is eventually a choice to be made by the import or export tool. This also enables modification of the existing *requirement hierarchy* instead of creating a new one every time the import functionality is performed.

The last three plug-ins of the UML AP Tool, org.topcased.modeler.requirementspecificationdiagram, org.topcased.modeler.controlstructurediagram and org.topcased.modeler.automationsequencediagram, implement the new diagram types of the automation profile: requirement specification diagram, control structure diagram and automation sequence diagram, respectively. The diagram types are registered to be used by the Topcased tool and graphical editors by defining an extension to the extension point org.topcased.modeler.diagrams (see Fig. 3). Extending this extension point includes specifying a diagram type specific configuration class and metaclasses that can act as root elements of the diagrams.

The first new diagram type, the requirement specification diagram, is aimed to enable modeling of automation application requirements and linking of

these requirements to actual design decisions. The diagram was implemented to resemble the class diagram, as can be seen in Fig. 4, since there wasn't any well known graphical representation of requirements in the automation domain to be used as a starting point.

The requirements in the Fig. 4, as well as modeling elements in figures 5 and 6, are related to a laboratory-scale process for simulating pulp batch cooking. The pulp process includes both continuous control tasks and discrete sequential control and is therefore a good example to be sketched with the new diagram types.

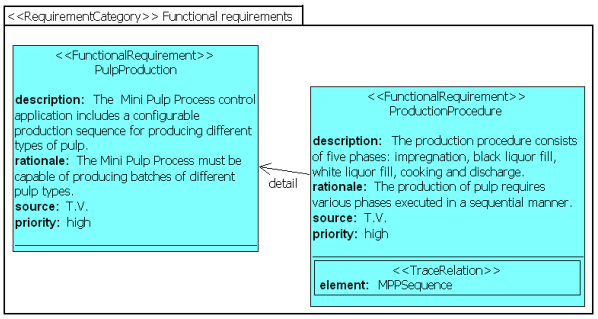

Fig. 4. Functional requirements of the Mini Pulp Process control application presented with the Requirement Specification diagram.

The second new diagram type, the control structure diagram, enables modeling of automation applications structure and control activities with use of function-block-like AutomationFunctions. In addition to AutomationFunctions, the diagram type allows the use of *SysML block definition diagram concepts* to enable modeling of AutomationFunctions coupling to the *whole control systems* and *I/O functionality*. An example of a control structure diagram can be seen in Fig. 5. AutomationFunctions defined by the automation profile are presented as blocks that can be linked together with ports and connections.

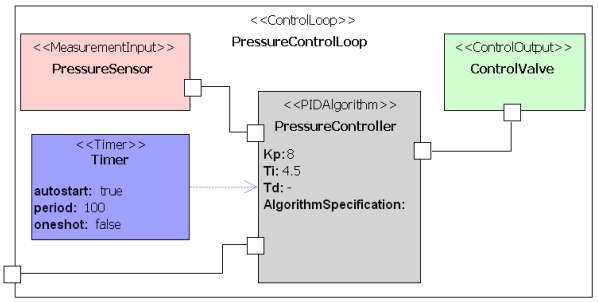

Fig. 5. Pressure control loop of the Mini Pulp process control application presented with the Control Structure diagram.

The third new diagram, the automation sequence diagram, enables modeling of sequentially executed *control activities*. Sequences consist of Steps, Interlocks (representing exceptional Steps), Allocations and sub-Sequence references that are executed in order determined by Transitions and Exceptions. The graphical presentation of the diagram type is mostly implemented to resemble traditional *sequential function chart* (SFC) diagrams of the automation domain. An example of such automation sequence diagram can be seen in Fig. 6.

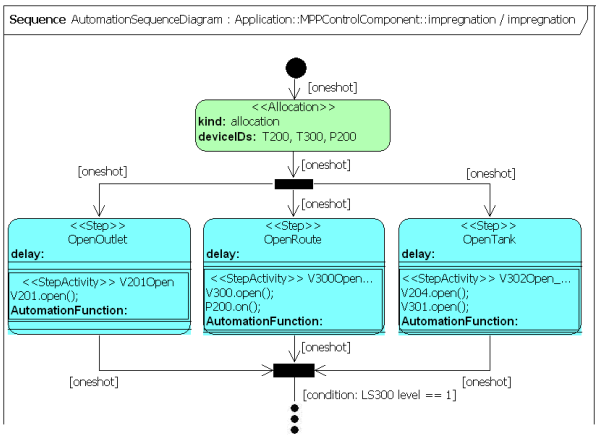

Fig. 6. Part of the impregnation sequence of the Mini Pulp Process control application presented with the Automation Sequence diagram.

7. Conclusions and further development

This paper has presented an overview of the UML automation profile and the AP-Tool. The former is a UML-based high-level modeling language for software-oriented applications in the automation domain, the latter is an Eclipse plug-in for supporting efficient use of the profile.

The current version of UML (UML V2) has sufficient potential for modeling tasks in the domain. Thus, the new language is implemented as an UML V2 profile which utilizes three existing UML profiles, namely SysML, UML Real-Time Profile and UML Quality of Service and Fault Tolerance Profile. The profile itself covers the most common essential aspects of automation applications, namely requirements, automation concepts, distribution and concurrency, as well as automation resources and device interfaces.

A model-driven approach for developing automation and control applications offers many possibilities to improve the development process. Although there are still issues that require consideration, this seems like a feasible approach that

is worthwhile studying. By bringing domain-specific concepts to the application design process problems can be solved by domain experts with familiar elements on a higher level compared to traditional approaches.

The implemented tool support for the UML automation profile proved it possible to extend UML on both M2 and M1 metamodeling layers and to implement new diagram types conforming to the extended metamodel with reasonable resources. The tool fulfills common functionalities of modeling tools and is technically implemented to utilize the plug-in architecture of Eclipse and various open source tools and frameworks, such as, the Eclipse Modeling Framework. The utilized plug-ins were found useful and well-working in their intended usage.

The prototype tool was also tested and evaluated by a few automation researchers from Tampere University of Technology and Helsinki University of Technology. According to the feedback, the new diagram types were considered fairly easy to draw and the tool performed modeling and drawing work rather well. Most of the difficulties that came to light were related to unfamiliarity of certain concepts of SysML and to the usage of UML profiles and stereotypes. It was also stated that the tool should include more user supporting functions as, for example, the use of stereotypes may be unfamiliar to automation and control engineers.

The AUKOTON approach and the tool chain presented is being further developed and evaluated in co-operation with our research partners and industry professionals. Although the tool chain has not yet been fully implemented and evaluated, it appears that the AUKOTON approach may provide improvements related to efficiency and productivity of automation application design and to quality and maintainability of automation applications.

The future work concerning the automation profile is focused on one hand to enhancing the Automation concepts subprofile with new algorithms and the DevicesAndResources subprofile with embedded device constraints. On the other hand, already in the ongoing work within the AUKOTON-project, the upper level requirements, given according to the new automation requirements standard (IEC-62424) will be integrated to the Requirements package of the automation profile.

Within the AUKOTON chain, partial code generation will be developed to facilitate run-time function block execution (IEC-61131) based on the specifications with the automation profile. Moreover, design wizards with smart design support functionality will be developed to help the use and dissemination of the profile within automation design community.

9. References

[1] AUTOSAR Gbr, UML profile for AUTOSAR V1.0.1, AUTOSAR Development Partnership, 2006.

[2] Object Management Group Inc., UML 2.1.1 Superstructure Specification, formal/07-02-05, OMG, 2007.

[3] T. Ritala, S. Kuikka, “UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry”, The Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria, 23-27.7.2007, pp. 885-890.

[4] C. Tranoris, K. C. Thramboulidis, “Integrating UML and the Function Block Concept for the Development of Distributed Control Applications”, The Proceedings of the 9th International Conference on Emerging Technologies and Factory Automation (ETFA 2003), Lisbon, Portugal, 16-19.9.2003, pp. 87-94.

[5] K.C. Thramboulidis, “Using UML in Control and Automation: A Model Driven Approach”, The Proceedings of 2nd IEEE International Conference on Industrial Informatics (INDIN 2004), Berlin, Germany, 24-26.6.2004, pp. 587-593.

Publication 2

Vepsäläinen, T., Sierla, S., Peltola, J., Kuikka, S. (2010) Assessing the Industrial Applicability and Adoption Potential of the AUKOTON Model-Driven Control Application Engineering Approach. Proceedings of the 8th IEEE International Conference on Industrial Informatics. Osaka, Japan, July 13-17, 2010, pp. 883-889.

DOI: 10.1109/INDIN.2010.5549626

Assessing the Industrial Applicability and Adoption Potential of the AUKOTON Model Driven Control Application Engineering Approach

Timo Vepsäläinen[1], Seppo Sierla[2], Jukka Peltola[2], Seppo Kuikka[1]
[1]*Tampere University of Technology, Department of Automation Science and Engineering,*
P.O. Box 692, FIN-33101 Tampere, Finland, {timo.vepsalainen, seppo.kuikka}@tut.fi
[2]*Aalto University, School of Science and Technology, Department of Automation and Systems Technology,*
P.O. Box 15500, FI-00076 AALTO, Finland, {seppo.sierla, jukka.peltola}@tkk.fi

Challenges in software development for automation and control have increased because of various reasons, including the size of applications and competition within the industry. A potential solution could be the use of a model-driven engineering approach to facilitate the development process and design information flow. Despite the wide interests towards model-based techniques in the automation domain, the authors have not been able to find a complete, tool supported UML based MDE approach, the industrial relevance of which would have been systematically assessed. In this paper, we intend to assess the suitability and usefulness of the AUKOTON development process and tools to development of industrial automation and control applications. To study these questions, we organized a laboratory project, in which industrial professionals used our development process and tools, and used qualitative research methods for gathering the industrial feedback.

I. INTRODUCTION

The present implementation techniques and practices of automation and control system implementations highlight the importance of software, i.e. the automation and control applications, as essential parts of the systems. Partly due to this, the challenges in software development for automation systems have increased. The systems have become more and more complex, while at the same time the competition and need for co-operation within the industry have become prevalent. This has forced the organizations to increase both the efficiency and openness of the development process.

A potential solution to the challenges could be the use of a high-level domain specific language accompanied with a model-driven engineering (MDE) approach and a tool environment supporting both of them. A development approach with such characteristics has been proposed in the AUKOTON project. The modeling concepts are based on the UML Automation Profile [1] defining the concepts for modeling the requirements, functionality and architecture of automation and control applications. The basics of the MDE approach, as well as the supporting tool environment, have been discussed in [2] and [3]. The latter paper also discusses the technical foundation for defining and implementing the transformations required by the development process.

However, adoption of MDE technology can be a difficult step for companies, especially since the scope of changes to current industrial development processes is unknown. This paper studies the adoption possibilities of the AUKOTON development process and tools and their ability to facilitate development work. A research methodology for involving professionals in this assessment is applied to this study.

This paper is organized as follows. Section 2 reviews research related to model-driven engineering of automation and control applications and presents our research questions. Research methodology for involving professionals and collecting empirical material is described in section 3. Current industrial development practices are discussed in section 4. Section 5 presents the case study and the gathered empirical material. The empirical material is analyzed against our research questions in section 6. Section 7 presents conclusions on the industrial adoption potential of the AUKOTON model driven engineering approach.

II. RELATED WORK AND RESEARCH QUESTIONS

The idea of model-driven engineering (MDE) and related approaches, e.g. Model-Driven Development (MDD), Model-Driven Software Development (MDSD) and Model-Driven Architecture (MDA), is to use models as primary engineering artifacts in the development process. In the process, models are gradually refined towards the executable application, either automatically and/or in a user-assisted manner. Model transformations are used to process models and to produce revised models from existing ones. Models, associated to model transformations and used during the development, usually conform to UML or some domain specific languages (DSLs) that can be either based on or independent of UML. Tool support for the processes exists on several platforms, such as Eclipse [16], for both the modeling [12], [13] and transforming [14], [15] purposes.

In addition to the AUKOTON project, the use of MDE in the automation domain has been recently proposed by several projects and papers. The approach of the MEDEIA project, as discussed in [17],[18] and [19], is based on Automation Components that are characterized to be composable combinations of embedded hard- and software including integrated simulation, verification and diagnostics services and which can be deployed to hardware with code generation. However, to the authors of this paper it remains unclear whether or not the maturity of the developed tools enable their practical assessment and whether or not the applicability

of the approach has been evaluated with industry professionals. Moreover, the focus of the project is in embedded control systems.

Model-driven development of distributed industrial control applications have been studied by Tranoris and Thramboulidis [20]. The proposed process addresses the requirements and analysis of industrial process measurement and control systems (IPMCS) using UML, and the design and deployment by means of the function block (FB) construct of IEC 61499. Model transformations are used between IPMCS-UML metamodel and IPMCS-IEC61499 function block metamodel, that both were defined in order to enable the use of model transformations and the development process is supported by the CORFU engineering support system. However, the authors of this paper are not aware of the development process and tools being assessed with industry professionals.

The suitability of modeling approaches and notations for model-driven development of industrial, distributed control systems has been studied in [21]. The evaluation of the modeling languages, which included mainly conventional modeling notations, such as, algebraic models, sequential function charts, IEC 61499 FBs and class and state diagrams, was based on modeling needs of the domain with respect to degree of formality, availability of analysis techniques and tools, expressiveness of requirements and availability of exchange formats. The evaluation, however, was not related to any specific MDE approach and the evaluation did not cover the industrial usefulness of any MDE approach per se.

Despite the wide interest towards model-driven methods in the domain, the authors have not been able to find any systematical assessment of industrial applicability of an MDE approach with capabilities to produce industrial automation applications and to automate part of the development work. The aim of this paper is to fill this gap by assessing the AUKOTON development process and UML AP Tool with domain-professional in the development of a small-scale industrial automation application.

Our research questions are 1) do the process and tool chain support the information content and development activities needed to produce industrial automation and control applications and 2) are the tools and the approach able to automate part of the development work that is done manually in current industrial practice. To study the questions, we organized a laboratory project in which industrial professionals used our process and tool chain to develop an application. Qualitative empirical material on their experience was gathered to answer the research questions above. Other MDE approaches on the domain, such as those cited above, were excluded from the study because of the tight schedule of the event and because the expertise of the authors that organized the event was in the AUKOTON development process and tools.

III. RESEARCH METHODOLOGY

A. Choice of method

For the purpose of assessing the industrial adoption potential of technology that is not yet in industrial use, such as the AUKOTON tool chain, most research methods are not applicable [4][11]. Surveys require considerable a priori knowledge of the phenomenon under study, so they are problematic for obtaining industrial feedback from technology that is not yet in industrial use. The designer of the survey would need to anticipate all the relevant factors and the answerers would need to foresee the difficulties and benefits of a technology that they are not familiar with [11]. Case studies can only be undertaken when a technology is being piloted in a company. This approach usually results in documenting innovations of industrial origin [4].

Few researchers have addressed the problem of evaluating DSLs and their supporting tools. A company pilot of a DSL is described in [24], but our goal is to evaluate a DSL with professionals before it gains sufficient maturity to be piloted. A solution to the problem of evaluating the performance of DSL models on various targets is described in [22]. This is an example of DSL evaluation that does not involve professionals. Experiences from having end users evaluate a DSL have been described in the context of the printing and graphics arts industry [23]; the paper only presents the results of the evaluation without describing the method for obtaining them, but it is clear from the results that the empirical material was qualitative. Qualitative methods are applicable for problems that have received little academic attention, so that the researcher is not able to specify the relevant variables beforehand [11]; since obstacles to industrial adoption of MDE-based control engineering approaches are not well known, qualitative methods were chosen in our research.

Since surveys, case studies and industry pilots are not applicable in our case, 8 professionals from 4 different companies were invited to participate in a one-day project using the AUKOTON tool chain on university premises. Research arrangements for this are described in the next section.

B. Research arrangements

Two methods are used for collecting empirical material in this research: participatory observation and interviewing. The arrangements are based on our experience in using these methods on two similar one-week projects in 2005 [10] and 2006 [9].

Participants use the AUKOTON development process and tools to develop a control system in a university PC-class. Appropriate guidance is provided in order to complete the small application with new tools during a one-day event. Guidance is given by researchers working in industrially familiar roles of lead engineers and technical specialists. These researchers have the dual role of collecting qualitative empirical material, and making field notes of problem situations in which their assistance is required. In this way, empirical material is focused on interesting problem situations; our earlier experience with the conventional participatory techniques was that most of the material simply confirms that participants were following instructions successfully [9].

With the research design described in this paper, researchers are continuously involved in recording situations in which participants need external help, and every such situation results in a qualitative field note. This arrangement is also very acceptable for industrial participants who can be uncomfortable having their activities observed and noted down, since the involvement of field-note taking researchers is visible to participants only as technical assistance.

One of our previous events [10] used recorded and transcribed semi-structured qualitative interviews according to the method proposed by [8]. Our discovery from the analysis of the transcripts was that despite the great volume of material and the significant effort in performing the transcripts, only a limited number of technically interesting observations were made by the interviewees. Since the participants had no previous experience with the tool, they were often not able to recall specific technical details that they had worked on only a few hours before the interview. In this situation, all the legitimate interviewing techniques, such as probing questions and follow-up questions [6] were of little use due to the fundamental problem of the interviewee not remembering sufficient details.

These problems were avoided with the interviewing strategy described here. A single interviewer is responsible for a small number of participants, and at the end of each phase in the workflow, a set of questions is asked from each of these participants regarding the workflow phase that was just completed. The questions concern the problem that the participant had recently worked on, and they can refer to the tool that is in front of them. The researcher will then note down compact detailed observations instead of producing hundreds of pages of interview transcripts from which only a minor part will be chosen for detailed analysis, as in conventional interviewing techniques [8]. This interview structure of asking the same questions after each workflow phase, and from each participant, results in a high level of comparability [6] in the material. Before the event, researchers visited each company for a full day interview about current industrial practice, which was structured according to the tasks in the development process and the related artifacts.

The resulting empirical material consists of qualitative notes collected by several methods: participatory observation, qualitative interviewing during the event and qualitative interviewing in companies before the event. The use of several empirical methods to study the same phenomena is advocated as it "will provide a richer picture of the events and/or issues than will any single method" [7, p. 163]. The methodological challenge is to link these three data sets together to support analysis. Our solution for this and the benefits of each of these 3 methods are described below.

Participatory observation complements the interviews during the event. In both cases, the researcher making the note also records the workflow phase and the name of the participant. Cross comparison of interview notes to any problem notes for the same person and phase suggest if interview statements were prompted by misunderstanding of the AUKOTON development process or a genuine understanding of the approach and its limitations. Interview statements were full of company specific jargon, so the meaning of terms could be discovered by referring to the task and activity descriptions of the corresponding company interview.

IV. INDUSTRIAL DEVELOPMENT PROCESS

This chapter describes the industrial design work flow and related design artifacts, based on interviews of six companies involved in process and automation design projects. The aim of the chapter is, thus, to position the control application engineering phase in industrial projects and to discuss the production of the design information that the AUKOTON development process and control application engineering phase in general use as source information.

Process control system design for process plants involves co-operation between teams and disciplines from process design, facility design, electrical design, instrumentation design and automation design. Instrumentation is often considered as a part of automation design. Automation design takes requirements and input information from the other mentioned areas of engineering and produces a functional process control application. In the Finnish process industry domain, the automation design is, typically divided in three main phases: preliminary design, basic design and detailed design. Depending on the project, each phase may be outsourced separately.

Main purpose of preliminary design is to provide the necessary information about major cost factors for making the investment decision. Thus, the phase resembles the feasibility study; however, the companies favor the term preliminary design. Similar phase is carried out also in other engineering areas. Preliminary automation design collects requirements and other source information from other engineering disciplines and the end customer. Customers may supply general design guidelines, standards and preferred device suppliers and technology selections. Process engineering provides a preliminary version of a P&I diagram, verbal process descriptions and a preliminary list of instruments (loop list). An updated loop list is received later from instrument designers. Electrical design provides a list of motors and other devices with significant power consumption. This enables estimation of required I/O points, which is used to evaluate the cost of the automation system. The structure of the automation system is documented in an automation description document. The produced artifacts are textual documents and updated spreadsheets, which provide source information for the basic design phase.

The basic design produces procurement specifications for the automation system. These are used when requesting quotations from automation suppliers. The I/O list is created on the basis of the process engineering's loop list, instrument list and motor list. When the automation system supplier is selected, more detailed technical information about the communication is accumulated in the I/O list. The Process engineering provides reference designations of the loops and functional descriptions of the process. An important workflow event is the control approach meeting between

process designers and basic designers. There may be several iterations of these meetings; first, emphasis is on understanding source information and later on presenting and validating basic design's design decisions and documents. The basic design produces design documents such as control diagrams and regulation diagrams (e.g. logic, interlock and sequence diagrams) and loop wise functional descriptions.

Control diagrams specify the analog control logic solution to be implemented, in order to manage the process in normal conditions and keeping it stable. One control diagram shows only a relevant section of the process equipment and its logical control solution using signal wires and calculation symbols. According to interviewees, control diagrams may use vendor neutral logic or vendor specific concepts if the automation system supplier has been selected. Regulation diagrams specify binary logic and sequential control for moving the process between operational states (e.g. start-up and shut-down of a sub-process), applying group control for several devices (e.g. synchronized operations) and handling abnormal conditions (e.g. interlocks). Similar to control diagrams, also regulation diagrams may use supplier specific or neutral logic and concepts.

V. THE ASSESSMENT

The activities performed during the assessment were divided to 6 phases: control approach discussion, source information inspection, requirement import and elaboration, functional design phase, platform specific design phase and PLCopen XML export. Model transformations were used for creating the requirement and functional models and to export PLCopen XML presentation of the application. Three of the phases: requirement import and elaboration, functional design phase and platform specific design phase, are integral parts of the AUKOTON development process presented in Fig. 1. The intention of the rest of the phases was to introduce the process to the developers and to be able to classify problem notes to more specific phases of the assessment event.

During the control approach discussion, the developers were introduced to the laboratory process, the source information documents and the approach for dividing the functionality to control loops. The information was presented by the lead engineers and consisted of a piping and instrumentation diagram (P&ID) presenting the physical process and spreadsheet documents of the I/O connections points and the required interlockings.

The laboratory process is presented in Fig. 2. The controlled process variables were the levels of water in tanks B100 and B200, water flow from tank B100 to B200, water temperature in tank B100 and pressure in tank B300. The control application was required to utilize both binary and analogue valued sensors and actuators, protect

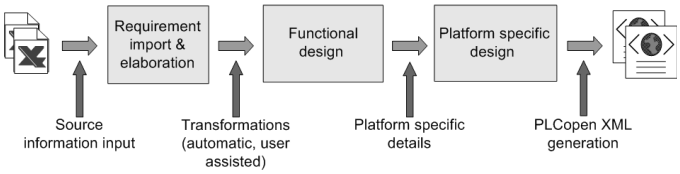

Fig. 1. An overview of the AUKOTON development process.

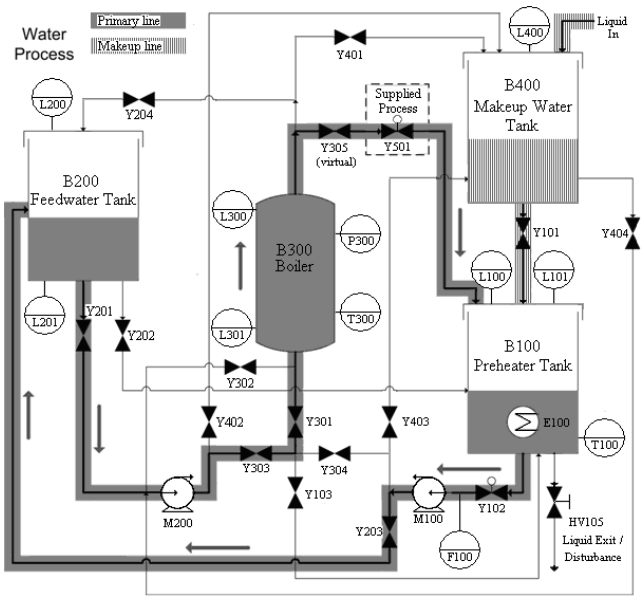

Fig. 2. The laboratory process automated during the assessment event.

instrumentation from misuse with interlocks and use controllers also in cascade control structures. Thus, the application covered the basic tasks of typical automation and control applications. Some of the interviewed developers were also able to identify a project with similar automated process from their work history.

After the discussion, the developers started working independently by continuing their study of the source material and then performing the actual development activities. In order to accomplish the development task during the one-day event, part of the required models and corresponding visual diagrams were given to the developers so that the diagrams could be used as parts of the design and as examples during the creation of the diagrams that the developers were responsible for. Guidance for the development approach and the tool were given as an instruction document and as help features integrated to the tool. The tool and its wizards partially automated some tasks, such as applying platform specific details to the models.

The AUKOTON tool chain produces application code in the PLCopen IEC 61131-3 XML format utilizing a pre-defined set of function blocks (FBs) called AUKOTON DCS (Distributed Control System) library. However, in addition to being able to produce IEC 61131-3 based PLC applications, an industrially applicable development approach should be able to support proprietary DCS platforms and their existing function block types. However, without resources to implement mappings for various target DCSs, the possibility to support various proprietary DCS platforms was assessed only based on interviews of the participants.

A. Requirement import and elaboration

The intention of the first actual development phase of the AUKOTON development process, requirement import and elaboration, is to import the source information and requirements to the tool environment. Source documents were imported as structured requirements presenting needs to interface with sensors and actuators of the process or to

compute control or interlocking signals, as presented in [3]. During the phase, the developers used two automatic import transformations to import the I/O connections points and the interlockings as system requirements, re-structured the requirements to new requirement categories, visualized the requirement model in *requirement specification diagrams* and added the requirements for controllers manually.

During the phase, a total amount of 20 problem notes, about 2/3 of the total number of problem notes, was gathered. The problems included, for example, developers forgetting to import a source information document or being unable to identify or find certain imported elements from the views of the tool. This could be due to several reasons. Firstly, current development practices of industrial automation applications do not usually contain phases corresponding directly to the requirement phase. Consequently, the developers may not have been able to identify a development phase of industrial projects that could be compared to the requirement phase in order to understand its purpose and the activities. Secondly, the requirement phase was the first phase during which the developers used the UML AP tool that most of the developers were not familiar with before the assessment. So, one factor behind these problems was the unfamiliarity of the tool and Eclipse platform. However, already during the functional design phase during which the actual activities with the tool were quite similar, such problems were avoided.

B. *Functional design*

The intention of the functional design phase of the development process is to produce a functional model of an application that fulfills the requirements and that could be later refined with platform specific details. During the phase, the developers used an automatic transformation for creation of the functional model (Automation Functions) based on the requirement model presenting the user-modified source information, visualized the functional model with *control structure diagrams* and could then refine and inspect the functional design. Automation Functions (AF) could be characterized as platform independent, abstract FBs representing different kinds of measurement, actuation, control and interlocking functionalities that can be combined and connected together for modeling of the whole application [3]. During the phase, only one problem note was reported and it reported difficulties to open a model.

Based on the interview notes gathered after the functional design phase, a majority of the developers agreed that the platform independent design could be completed for, at least, some proprietary DCS platforms. However, it was pointed out that more specific concepts, such as logic operations allowing the modeling of interlocking logic, would be needed to specify interlockings in industrial projects. This information was suggested to be added in diagrams specifying the inner logic of the interlocking Automation Functions (see [1]). Another piece of information that would be needed for certain platforms was the specification of the execution order of the AFs that are eventually transformed into FBs. In case of PLCopen IEC 61131-3, this information was not needed because the Multiprog tool could decide the order after the code generation.

Some developers also suggested presentation of control and interlocking functionalities in different diagrams in order to aid the understandability of the design. Technically, such views could be created with the tool but creation of them is not automated and they were not used during the assessment. However, the separation of such aspects might be beneficial already during the requirements specification phase. In addition, some developers suggested improving the graphical presentation of the models. Vocabulary of the model elements, as well as graphical presentation, should be more familiar to automation engineers and the UML specific details and vocabulary should be hidden from the developers.

C. *Platform specific design and PLCopen export*

The intention of the platform specific design phase is to produce a platform specific model based on which the application code could be automatically generated. During the phase, the platform independent functional model was completed with platform specific (AUKOTON DCS library) stereotypes and ports. The stereotypes were used to map the AFs to existing FBs so that modeling of the implementation of the AFs was not needed. The signal interfaces of the AFs (presented as ports) were completed to correspond to those of the existing FBs by (automatically) checking that all the ports of the FBs were present in the model and by warning about additional ports not defined by the actual, implementing FBs.

The interview notes gathered after the completion of the phase supported our understanding that the stereotypes and ports based approach could be used also for completing the platform independent design for certain proprietary DCS platforms. However, the possibility for detailed-enough specification of the interlockings was still seen to be missing which could reduce the usability of the approach. The missing details included priorities, delays and detailed specification of the inner logic of the interlocking Automation Functions. It was also estimated that the basic control functionality (without interlockings) forms approximately 80-90% of the amount of code in typical customer projects.

Finally, the developers performed the automatic code generation and could visualize the program with Multiprog, a tool with PLCopen IEC 61131-3 XML import/export capability. No problem notes were reported during the code generation.

VI. RESULTS

This section presents the results of the assessment based on the problem and interview notes gathered during the event. The discussion has been structured based on the research questions presented in section 2. Subsection A discusses the question of whether the process and tool chain enable the development of industrial automation applications for proprietary DCS platforms. The discussion will consider both the developers' ability to use the development process and tools and the possibility to use this MDE technique to generate applications to the proprietary DCS platforms that are currently used by automation system vendors.

Subsection B addresses the second research question - whether the approach and the techniques could automate part of the development work that is usually done manually in

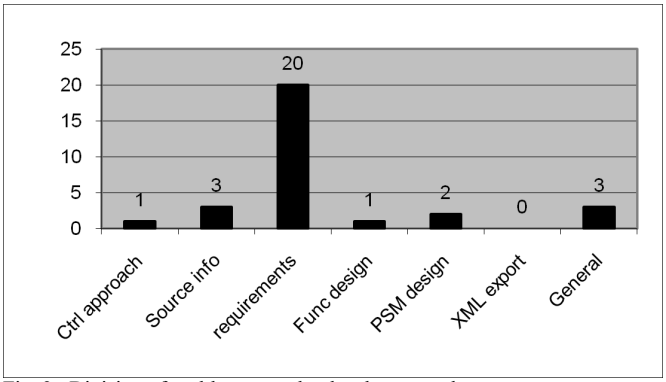

Fig. 3. Division of problem notes by development phase.

current industrial practice. However, also in case of the second research question, the ability to automate part of AUKOTON development process may not automatically prove the industrial capabilities. Instead, in order to assess industrial applicability, aspects such as inconsistencies between industrial development workflow and that of the AUKOTON development process should be taken into consideration. However, because of the size of the developed application, such aspects can only be assessed based on the results of interviewing industrial professionals.

A. Ability to produce industrial automation applications

During the development activities, a total of 30 problem notes were gathered. The numbers of problem notes gathered during the phases of the assessment are: control approach discussion (1), source information inspection (3), requirements import and elaboration (20), automation functions (1), platform specific design (2), PLCopen XML export (0). In addition, 3 problem notes were gathered that cannot be linked to any specific development phase. Fig. 3 presents the distribution of problems based on the development phases.

As presented in previous section, at least some of the problems during the requirements phase may have been caused by the unfamiliarity of the tool and the platform. This assumption is also supported by the division of problems based on the types of problems, shown in Fig. 4. For example, the total amount of problems related to finding and creating elements and using the Eclipse views were 2, 3 and 3, respectively. Problem notes in these categories reported

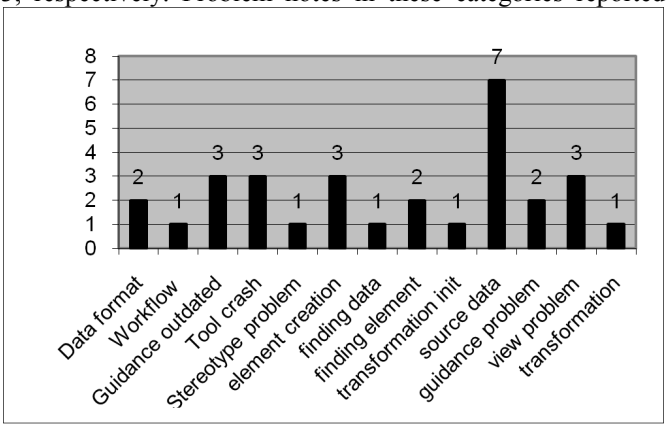

Fig. 4. Division of problem notes by types of problems.

difficulties to find or notice elements and element creations in the outline view of the tool. In addition, some problem notes reported difficulties to joint-use the graphical diagrams and the outline view, and to comprehend the correspondance between the elements of the views. Because such features are quite common in UML tools, those tools and the Eclipse platform have probably been unfamiliar to the developers. In addition, the distribution in Fig. 4 indicates difficulties interpreting source data. However, 6 of those 7 problem notes were more like comments regarding the laboratory process and its documentation, such as, presentation of the cascade control loop in the P&ID.

As the total amount of developers was eight, the average sum of problem notes per developer is approximately four that could be considered relatively small. In addition, a majority of the developers were able to finish the development and perform code generation during the one-day event. The interview notes gathered after the platform specific modeling phase supported our understanding that proprietary DCS platforms with existing collections of FBs (excluding interlocks), that have been used successfully in industrial scale projects, could be supported with the process and tools. Thus, although the Eclipse platform and UML based modeling may be unfamiliar to automation professionals, such techniques and tools could possibly be successfully used in automation application development.

B. Ability to automate part of the development work

The question of whether the approach can automate some development tasks that are carried out manually in current practices proved to be more complicated. Based on the interview notes, the approach was seen beneficial as "the current industrial practice can be seen as collecting pieces of design information and acting on unstable information". The AUKOTON development process, on the other hand, was seen to proceed in a more straightforward way from abstract levels to more concrete design. The most important improvement offered by the approach was the partially automated shifting from process and basic automation design based requirements to application development. However, somewhat similar intermediate techniques, such as, spreadsheet macros are already in use to automate, for example, parameter updates in design. Secondly, it was seen that transformation techniques could be used for automating information transfer between development phases in order to reduce both workload and errors.

Based on the interviews, the ability to partially automate the development may not be automatically generalizable to the present development practices used within the industry. In case of the AUKOTON development process, the automated creation of functional model structures was made possible by detailed requirement specification models; the creation of such models would require additional work compared to the current development practices. Based on the interviews, the information content of the requirement models is gathered also in the present practices but not as formally and not necessarily during an individual development phase dedicated for the requirements as in case of the AUKOTON process.

Instead, according to an interviewee, in their development practice, the source information from process engineering is used for creation of control, interlocking and regulation diagrams that are used as inputs for the application development. As such, the requirement models of the AUKOTON approach does not correspond to any of the aforementioned diagrams but to unified version of them. An interviewee suggested providing control and interlocking views to this unified model. With such an approach, the requirement model within the tool could still be as unified as in the case of the present approach although different views would be used for depicting, for example, the control and interlocking aspects.

VII. CONCLUSIONS

As a conclusion to the first research question, the results of the development task and the interviews suggest that the approach could be used for development of industrial DCS-based applications. Three areas of further development were identified. Firstly, the capability to specify interlockings should be enhanced. Secondly, UML related terminology and notations should be replaced with terms and symbols that are familiar to professional automation engineers. Thirdly, visualization and editing of designs at different levels of granularity is needed.

As a conclusion to the second research question, the interviews suggest that some design activities of present development practices could be automated but with the cost of introducing an additional but at least partially automated work phase with requirement models. In addition, industrial application of the development approach might require changes to industrial development practices. Additional activities would be required especially for unifying the source information in order to create a starting point for the application development. Nevertheless, the total amount of development work could still be smaller because also the present development practices require unification of the design later during the application development. Estimating the quantitative net effect on development work, however, requires further research. Comparisons to industrial practice are only meaningful after the MDE tools mature to commercial quality. In addition to the AUKOTON project, model-driven methods have been recently proposed by several projects in the automation domain. Despite the process and tools used during the reported assessment were specific to the AUKOTON project, the authors hope that part of the industrial feedback could be of use also for the other projects.

REFERENCES

[1] T. Ritala, S. Kuikka, "UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry", The Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria, 23-27.7.2007, pp. 885-890.

[2] Vepsäläinen, T., Hästbacka, D., Kuikka, S. Tool Support for the UML Automation Profile - for Domain-Specific Software Development in Manufacturing. In The Proceedings of the 3rd International Conference on Software Engineering Advances (Sliema, Malta, 26-31 October, 2008). ICSEA'08. IEEE Computer Society, Washington, DC, USA, 43-50.

[3] Vepsäläinen, T., Hästbacka, D., Kuikka S. A Model-driven Tool Environment for Automation and Control Application Development - Transformation Assisted, Extendable Approach. In The Proceedings of the 7th Nordic Workshop on Model Driven Software Engineering (NW-MODE'09), Tampere, Finland, August 26-28, 2009.

[4] Benbasat, I., Goldstein, D.K., & Mead, M. (2002) "The Case Research Strategy in Studies of Information Systems". In Myers, M.D. and Avison, D. (eds) Qualitative Research in Information Systems. London: SAGE Publications, pp 79-100.

[5] Browne, G.J., & Menon, N.M. (2004). Network Effects and Social Dilemmas in Technology Industries. IEEE Software 21(5), 44-50.

[6] Kvale, S. (1996). Interviews: an introduction to qualitative research interviewing. Thousand Oaks, CA: SAGE Publications.

[7] Sawyer, S. (2001) "Analysis by Long Walk: Some Approaches to the Synthesis of Multiple Sources of Evidence". In Trauth, E.M. (ed) Qualitative Research in IS: Issues and Trends. Hershey PA: Idea Group Publishing, 163-190.

[8] Seidman, I. (1997). Interviewing as Qualitative Research: A guide for Researchers in Education and the Social Sciences. Second Edition. New York: Teachers College Press.

[9] Sierla, S.A., Christensen, J.H., Koskinen, K.O., & Peltola, J.P. (2007). Educational Approaches for the Industrial Acceptance of IEC 61499. ETFA'2007 12th IEEE International Conference on Emerging Technologies and Factory Automation, September 25-28, 2007, Patras, Greece.

[10] Strömman, M.P., Sierla, S.A., Peltola, J.P., & Koskinen, K.O. (2006). Professional designers' adaptations of IEC 61499 to their individual work practices. ETFA'2006 11th IEEE International Conference on Emerging Technologies and Factory Automation, September 20-22, 2006, Prague, Czech Republic.

[11] Trauth, E.M. (2001) "The Choice of Qualitative Methods in IS Research". In Trauth, E.M. (ed) Qualitative Research in IS: Issues and Trends. Hershey PA: Idea Group Publishing, 1-19.

[12] Papyrus UML project, http://www.papyrusuml.org

[13] Topcased project, http://www.topcased.org/

[14] SmartQVT project, http://smartqvt.elibel.tm.fr/

[15] ATL project, http://www.eclipse.org/m2m/atl/

[16] Eclipse project, http://www.eclipse.org/

[17] Strasser, T., Sunder, C., Valentini, A. Model-driven embedded systems design environment for the industrial automation sector. INDIN 2008 the 6th IEEE International Conference on Industrial Informatics, July 13-16, 2008, Daejeon, Korea.

[18] Strasser, T., Rooker, M., Hegny, I., Wenger, M., Zoitl, A., Ferrarini, L., Dede, A., Colla, M. A research roadmap for model-driven design of embedded systems for automation components. INDIN 2009 the 7th IEEE International Conference on Industrial Informatics, June 23-26, 2009, Cardiff, UK.

[19] Ferrarini, L., Dede, A., Salaun, P., Tuan Dang, Fogliazza, G. Domain specific views in model-driven embedded systems design in industrial automation. INDIN 2009 the 7th IEEE International Conference on Industrial Informatics, June 23-26, 2009, Cardiff, UK.

[20] Tranoris, C., Thramboulidis, K. (2006). A tool-supported engineering process for developing control applications, Computers in Industry, Vol. 57, pp.462–472.

[21] Luder, A., Hundt, L., Biffl, S. On the suitability of modeling approaches for engineering distributed control systems. INDIN 2009 the 7th IEEE International Conference on Industrial Informatics, June 23-26, 2009, Cardiff, UK.

[22] Zhu, L., Liu, Y., Bui, N.B., Gorton, J., Revel8or: Model Driven Capacity Planning Tool Suite, ICSE 2007 29th International Conference on Software Engineering, 20-26 May 2007, Minneapolis, USA, Page(s):797 – 800.

[23] Dantra, R.; Grundy, J.; Hosking, J., A domain-specific visual language for report writing using Microsoft DSL tools, VL/HCC 2009. IEEE Symposium on Visual Languages and Human-Centric Computing, 20-24 Sept. 2009, Corvallis, USA Page(s):15 – 22

[24] Lee, J.-S.; Chae, H.S., Domain-specific language approach to modelling UI architecture of mobile telephony systems, Software, IEE Proceedings, Volume 153, Issue 6, Dec. 2006 Page(s):231 - 240

Publication 3

Vepsäläinen, T., Kuikka, S. (2014) Integrating Model-In-the-Loop Simulations to Model-Driven Development in Industrial Control. SIMULATION: Transactions of the Society for Modeling and Simulation International, vol. 90, no12, pp. 1295-1311.

DOI: 10.1177/0037549714553229

Publication 4

Vepsäläinen, T., Kuikka, S. (2014) Model-Driven Development of Automation and Control Applications - Modeling and Simulation of Control Sequences. Advances in Software Engineering. Hindawi Publishing Corporation. Available: http://www.hindawi.com/journals/ase/2014/470201/

DOI: 10.1155/2014/470201

Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2014, Article ID 470201, 14 pages
http://dx.doi.org/10.1155/2014/470201

Research Article

Model-Driven Development of Automation and Control Applications: Modeling and Simulation of Control Sequences

Timo Vepsäläinen and Seppo Kuikka

Department of Automation Science and Engineering, Tampere University of Technology, P.O. Box 692, Korkeakoulunkatu 3, 33101 Tampere, Finland

Correspondence should be addressed to Timo Vepsäläinen; timo.vepsalainen@tut.fi

Received 20 March 2014; Revised 24 June 2014; Accepted 8 July 2014; Published 7 August 2014

Academic Editor: Henry Muccini

The scope and responsibilities of control applications are increasing due to, for example, the emergence of industrial internet. To meet the challenge, model-driven development techniques have been in active research in the application domain. Simulations that have been traditionally used in the domain, however, have not yet been sufficiently integrated to model-driven control application development. In this paper, a model-driven development process that includes support for design-time simulations is complemented with support for simulating sequential control functions. The approach is implemented with open source tools and demonstrated by creating and simulating a control system model in closed-loop with a large and complex model of a paper industry process.

1. Introduction

Model-driven development (MDD) is a system and software development methodology that emphasizes the use of models during the development work. In MDD, models conform to modeling languages that have formal metamodels, for example, unified modeling language (UML). In addition to manual development work, models can be processed with model transformations that revise existing and create new, refined models. The use of transformations may automate error-prone tasks such as importing information to models from preceding development phases and tools. Design models can be used for generating code or to analyze the developed systems. Automated model checks may reveal problems and inconsistencies in models and between phase products.

The mentioned benefits of MDD are related to development tasks that are repetitive and simple enough to be treated with preprogrammed rules. However, MDD has not been able to, and probably cannot, automate all the complex tasks in system and software development. Demanding design decisions over alternative solutions to achieve (sometimes informal) objectives and product characteristics need to be made by professional developers. However, although genuine design decisions cannot be automated, developers do not always have to rely solely on their experience. For example, simulation is a technique that has been traditionally used in the domain within control algorithm development and control system testing.

Automation and control system development is also an application domain in which the use of MDD techniques has been researched extensively during recent years. However, despite the research activities and the tradition of using simulations, ability to simulate early software design models has not yet been sufficiently addressed in the domain.

In their previous work, the authors have developed a simulator integration [1] to the tool-supported Aukoton MDD process [2] for automation and control applications. The approach is based on UML Automation Profile (UML AP) [3]. It enables modeling and simulation of cyclically executed control functions including feedback and binary control as well as interlocks (interlocks are used in control systems to protect the controlled processes from causing harm to themselves or personnel, e.g., by forcing actuators to safe states based on measured states of the processes).

The simulation support is intended to be usable during both platform independent and platform specific modeling

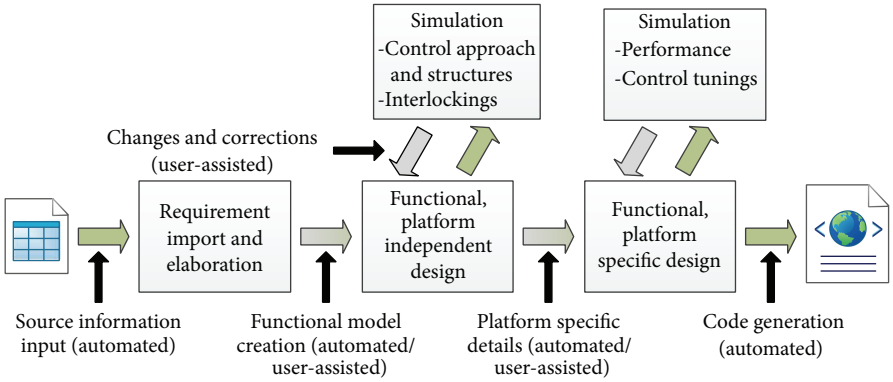

Figure 1: The MDD process with simulation extensions.

phases of the development process; see Figure 1. During the platform independent phase, it is possible to, for example, evaluate alternative control approaches, structures, and interlocks. During the platform specific phase, the approach enables the evaluation of platform specific functions, tunings, and predicted overall performance of the system. Technically the approach utilizes model-in-the-loop simulations so that UML AP control system models are transformed to ModelicaML models. In this paper the approach is complemented by enabling simulation of sequential control activities. The activities are modelled with Automation Sequences of UML AP and visualized with Automation Sequence Diagrams.

The contributions of this paper are as follows. The modeling notation is discussed in comparison to the well-known state machine notation of UML. An approach that enables the simulation of control sequences in a state-machine-like form is presented and implemented as a model transformation. The approach is integrated to the previous simulation integration. The approach is applied to batch control of a paper industry process.

The rest of this paper is organized as follows. Section 2 reviews work related to the use of MDD and simulations in the industrial control domain. In Section 3, the previous work is introduced briefly, which is necessary for understanding how the new work integrates to it. Section 4 discusses the use of control sequences in process industry, presents the UML AP approach to modeling control sequences, and presents the model transformation to create simulation models. In Section 5, before discussion and conclusions, the approach is applied to an illustrative pulp batch processing system.

2. Related Work

The use of model-driven techniques has been researched extensively in the domain of industrial control during recent years. Modeling of requirements, architecture, and details of control applications has been seen as an important part of design processes and as a means to cope with the ever-increasing size and complexity of the applications. Many of the recent approaches have also integrated simulations to the development processes in order to be able to test early and concurrently to the development work.

In addition to industrial control, MDD with simulation features has been applied to control system development for automotive and other embedded applications. The general simulation approaches that can be applied when models are used for generating code include model-in-the-loop (MiL), software-in-the-loop (SiL), processor-in-the-loop (PiL), and hardware-in-the-loop (HiL) simulations [4]. The approaches differ in the control system configurations that are used to control plant models in closed-loop simulations. Examples on use of MiL, SiL, and HiL simulations in the embedded system domain include [5] that describes a general framework for and two examples of use of MiL simulation. A testing environment that uses SiL simulations is presented in [6]. HiL simulation and testing have been utilized, for example, in [7–9].

Another classification of simulation approaches is related to the amount of simulation engines. Simulation of a controlled system, with a model of a process to be controlled and a model of a control system, can be performed within a single simulation engine or as cosimulation. In cosimulation, the models are simulated within different but connected environments. This requires a mechanism to synchronize the simulations including their values and states. Commands and functions, for example, running, replaying, freezing, and loading states (see [10] for a list of basic simulation functions) must be replicated to all used engines.

The management and coupling of cosimulations have been recently addressed with FMI standard [11] and also with model based techniques [12]. However, the area of expertise of control application developers may still not be in simulation techniques. As a consequence, the use of a single simulation engine can be considered more recommendable.

In the industrial control domain, MAGICS approach for MDD of industrial process control software is presented in [13]. As a modeling notation the approach utilizes ProcGraph that has been implemented on the Eclipse platform on top of Eclipse Modeling Framework (EMF). The approach utilizes several diagram types including Entity Diagrams (ED), State Transition Diagrams (STD), and State Dependency Diagrams

(SDD), of which STD is suitable for modeling sequential behavior. The approach enables the generation of executables but does not address simulations.

The FLEXICON project studied the integration of Commercial-Off-The-Shelf tools, including MATLAB/Simu- link and ISaGraf, to support the development of control appli- cations for marine, automotive, and aerospace systems [14]. The approach uses cosimulation, which is enabled by DSS (data delivery service) middleware between the tools.

Vyatkin et al. [15] developed a model-integrated design framework for designing and validating industrial automation systems. It is based on the Intelligent Mechatronic Component (IMC) concept and the use of IEC 61499 architecture. New systems are developed from IMCs that are integrated together and with their models enable formal verification, closed-loop MiL simulation of IEC 61499 models and code deployment.

The approach of the MEDEIA project [16] builds on use of several model types as well as bidirectional model transformations. The process supports the use of closed-loop MiL simulations which are based on use of an IEC 61499 environment. Simulation models of the process parts are in the approach defined with either timed state charts or external behavior descriptions (external simulation tools).

The abovementioned standard, IEC 61499 [17], is a specification and modeling language for industrial control applications. It extends the function block concept of another IEC programming language, IEC 61131-3 [18], with event driven execution and support for distribution of applications. With an appropriate tool support, IEC 61499 models can also be used for simulation purposes.

The simulation approach in [19] is based on mixing real control hardware with simulated one while simulating the plant in another (SIMBA 3D) environment. The benefit of the cosimulation approach is the ability to test early, by executing already implemented parts and simulating the rest.

The simulation approach closest to our work has been recently presented in [20]. In a manner similar to [15, 16] IEC 61499 is used for simulation purposes also in [20]. Model transformations are used for creating IEC 61499 plant models from MATLAB/Simulink plant models to obtain closed-loop behavior within a single (MiL) simulation environment.

Difference from the work to be presented, the referred simulation approaches in the domain ([14–16, 19, 20]) do not address modeling and simulation of sequential control separately from, for example, stabilizing feedback control. In [13] the sequential control aspect is addressed with respect to modeling. However, simulation of the models is not suggested. The use of simulations can thus be assumed to be possible no earlier than after code generation.

On the other hand, the referred development approaches that support simulations rely either on cosimulation ([14, 19]) or use of IEC 61499 as a simulation language ([15, 16, 20]). The use of IEC 61499 to simulations was not a viable alternative in this work to be presented because it is not used in the MDD process [2] that is extended with simulations. On the other hand, UML AP models that are used in the development process are not simulatable as such. The use of cosimulation would thus have either required a transformation to a simulatable form or delayed simulations to simulating plant models with produced executables. These reasons, however, apply to a number of MDD processes with nonsimulatable modeling languages such as UML and UML profiles.

Other works related to sequential control with model-based characteristics include [21] that presents an approach to transform Grafcet [22] models to Mealy machines for testing purposes (Grafcet is a conventional means to specify control sequences). Execution semantics of Sequential Function Charts (SFC) [18] have been addressed in [23]. The SFC notation is part of IEC 61131-3 [18] and based on the earlier version Grafcet.

In the simulation approach to be presented, the target simulation language is Modelica [24] with Modelica Modeling Language (ModelicaML) [25] as an intermediate language. Modelica is an object-oriented, equation based simulation language. The basic concepts of it are simulation classes that contain properties, equations, and connectors. Similarly to classes of object-oriented programming languages, Modelica classes can inherit properties (and equations) of parent classes. Simulatable Modelica models consist of instances of the classes that are connected together with their connectors. ModelicaML [25], on the other hand, is a UML profile for Modelica. It consists of stereotypes and tagged values that correspond to the key words and features of Modelica and enable modeling of Modelica models with UML tools. ModelicaML models are not simulatable as such but can be transformed to simulatable Modelica form with OpenModelica tools [26]. For simulating Modelica models there are both open source (e.g., OpenModelica [26]) and commercial tools (e.g., Dymola [27]) available.

The ModelicaML (profile) implementation uses UML2 plugins on the Eclipse platform which are built on Eclipse Modeling Framework (EMF) implementation of OMG Meta Object Facility (MOF). The ModelicaML implementation is thus technically similar to the UML AP implementation [28] that is used in this work for control system modeling. It has been implemented by extending UML2 and Topcased SysML metamodel with EMF. This similar background of the tools enables implementing the transformation from UML AP to ModelicaML using standardized QVT (Query/View/Transformation) languages [29] and their open source implementations on the Eclipse platform.

3. On Simulating Control Application Models

The objective of integrating simulations to MDD for automation and control applications is to support design-time quality assurance activities. It should be possible to compare alternative control approaches and structures and tunings as well as interlocks. Design flaws should be found and corrected as early as possible and to the extent possible so that they would not affect adversely subsequent design phases. By enabling simulation of design-time models, it could be also possible to obtain at least part of the general benefits of simulations before implementation of the applications. Such general benefits include, for example, improvements

to the design, development, and validation of the control programs, as reported in [10].

Without specific support for sequential control, the approach to create simulation models from UML AP models has been presented in [1]. In UML AP, the modeling concepts for functional modeling are automation functions (AFs) that have been divided to a hierarchy of measurements, actuations, controls, and interlocks. Measurement and actuation AFs are interfaced with sensors and actuators of the controlled processes while performing conversions of signals to and from engineering units. Control AFs perform computation of control signals according to control algorithms. The purpose of interlock AFs is to compute releasing and locking signals for actuators and devices. AFs interchange signals and information with ports.

The transformation for simulating functional UML AP models (that consist of AFs) creates and appends simulation counterparts of the AFs to Modelica plant simulation models. For platform independent AFs, the transformation utilizes a library of predefined simulation counterparts (classes) of them. To support platform and vendor specific AFs, the transformation is capable to utilize external libraries of simulation classes. To support application specific AFs, for example, interlocks that require tailoring for each application, the transformation is capable to create simulation classes based on logic diagram descriptions of AFs [1]. The process described in this paper is an equivalent approach to create new simulation classes but based on Automation Sequence Diagrams instead of logic diagrams.

The decision to use model transformations in this manner was made because UML AP models, as they are used in the tool, are not simulatable as such. Transforming plant models to the control application models would not have enabled closed-loop simulation, for example, in [16, 20] in which (IEC 61499) models were simulatable. In a similar manner, use of cosimulation as in [14, 19] would have required transformation to a simulatable form before applying cosimulation. Additionally, the cosimulation approach would require additional work; see Section 2, related to, for example, coupling simulations skills that not all control application developers can be assumed to have. The approaches to obtain closed-loop simulations by transforming plant models, by transforming control application models, and by using cosimulation have also been recently compared in [30].

An example structure of a plant model before and after executing the transformation that appends the control application specific parts to it is illustrated in Figure 2. Before executing the transformation, the model contains simulation class definitions of the parts of the plant and a description of how the interconnected instances of the classes form the system model. This part of the model, referred to as the original process model, is circled with blue, dashed line. The transformation (1) copies and creates new simulation class definitions based on the control system model, (2) creates instances of the classes according to the control system model, and (3) couples the required instances of the classes to the original model. In the figure, the newly created parts of the model are circled with red, dashed line.

4. Modeling and Simulation of Control Sequences

Control sequences are needed by process industries to perform start-ups of complex processes, for example, power plants or paper machines and to drive them to their designed operating states. In a similar manner, shutting down a process in a controlled and energy efficient manner may require changing set-points of process variables and shutting down devices and sub-systems in a specific order. On the other hand, batch processes constitute a challenging part of industrial processes. In batch processes production of the end products may require, for example, addition of source materials and substances according to time constraints and achievement of defined process states, for example, temperatures and concentrations.

The UML AP approach to modeling sequential control is based on (automation) sequences that have been developed to enable a SFC-conformant modeling notation within UML AP models. Sequences are modelled with a domain specific, new diagram type, Automation Sequence Diagram (ASD). Graphically the ASD notation resembles both the state machine and activity modeling notations of UML.

4.1. Description of the Modeling Notation. Sequences that are described in ASDs consist of Steps that are basic procedural elements in the approach (e.g., upper level batch recipe steps or device level controls). Similar to states of UML state machines, Steps contain Entry, Step, and Exit Activities that are executed when arriving to the Step, during the Step and when exiting the Step, respectively. In addition, Steps may reference other Sequences that can be defined with other ADSs. This is an equivalent characteristic to composite states of UML. Containing activities and referencing a sub-Sequence are exclusive alternatives for a Step. In addition to basic Steps, Sequences may contain Allocations. Allocations are intended for reserving process items and devices for the Sequences that they appear in. When used, Allocations are next to initial Steps in the Sequences.

The execution order of Steps within a Sequence is determined by Transitions that may contain different kinds of conditions that control when a Transition is fired. First, the condition can be a Boolean condition that explicitly specifies a Boolean valued condition based on, for example, values of the variables of the AF that contains the Sequence. Secondly, a condition can be a timeout condition specifying how long the Transition must wait after the execution of the previous Step is finished. Additionally, the Transition can be a one shot Transition which is fired immediately after the previous Step has been executed.

In addition to Steps, Allocations, and Transitions, Sequences contain initial and final as well as fork and join Steps. They can be used in a similar manner that the corresponding pseudostates of UML state machines, that is, to control the execution of Sequences. Use of initial and final Steps is also a necessity in each Sequence because whether a transition may occur from a Step to another is not always dependent (only) on the conditions of the Transitions.

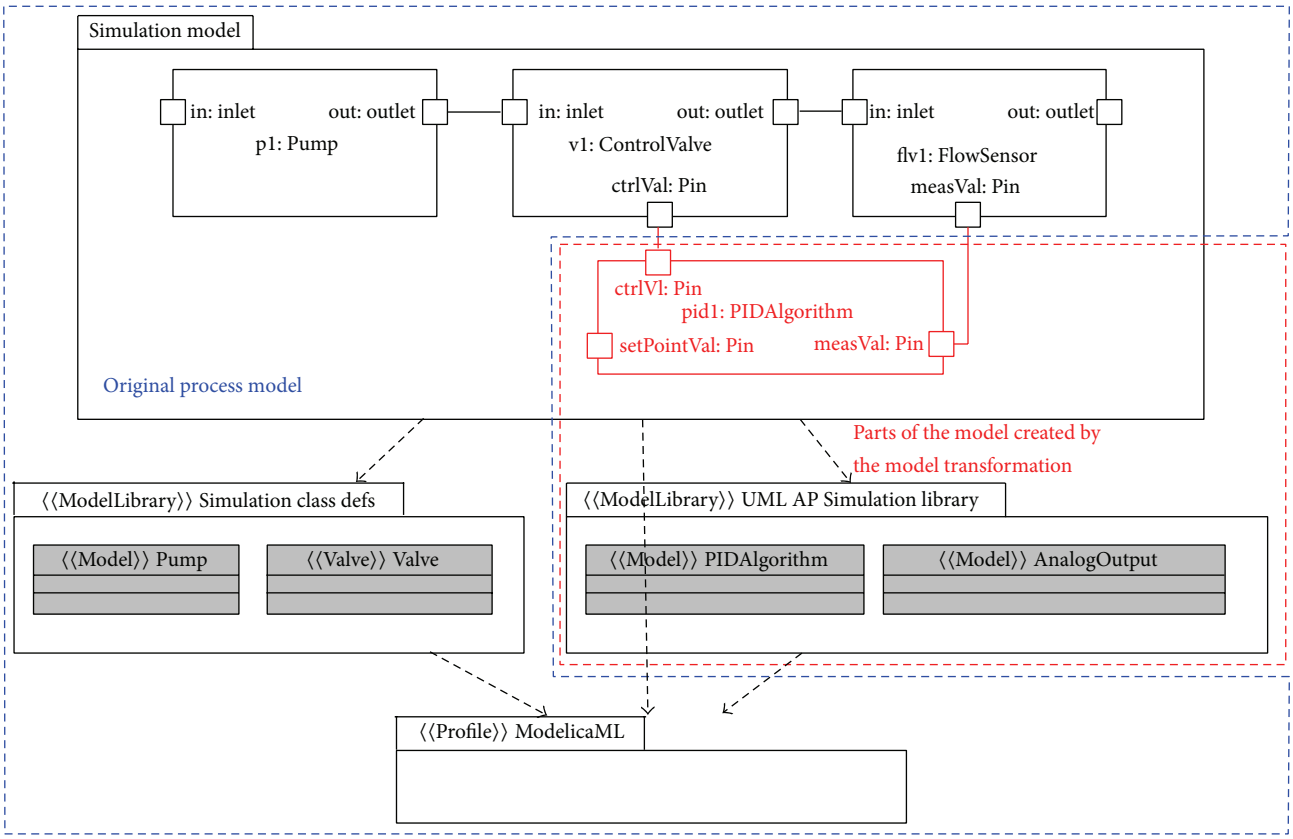

Figure 2: The transformation adds the control system specific parts to an existing model of the physical process.

Consider, for example, the two example diagrams in Figure 3. The figure also illustrates the graphical representation of initial and final Steps, (basic) Steps, Steps that reference sub-Sequences, forks, joins, and Allocations. In the Sequence at the left-hand side, all the Steps reference sub-Sequences that consist of Steps and possibly other sub-Sequences. For example, the WLF (White Liquor Fill) Sequence in the right-hand side diagram is referenced from the third Step in the left-hand side diagram.

Because the Transitions in the Sequence at the left-hand side are one shot Transitions, it is obvious that whether a Transition can fire is also dependent on the completion of the referenced Sequences. Referenced Sequences need to have performed their control activities in a similar manner as in SFCs [18], which is a domain specific notation based on which the ASD notation has been developed. For a Transition to be fired from a Step referencing a sub-Sequence, the referenced Sequence must have reached its final Step. This is a clear semantic difference of the notation in comparison to UML state machines.

Some other obvious differences to UML state machines are also visible in Figure 3. The first Step in the Sequence on the right-hand side is an Allocation. In the example, the allocated process parts are (tanks) T100, T300, and T400 as well as (pump) P100. (In UML state machine diagrams, there are no similar concepts.) After the first of the fork Steps, the transition condition on the right-hand side is of type timeout with value "1" indicating that the Transitions must wait 1 (sec) after the execution reaches the fork. In UML state machines the semantics of the timeouts is slightly different since the waiting time of UML AP Transitions starts from the completion time of the Step preceding the transition.

Lastly, as can be seen in the example Sequence at the right-hand side in Figure 3, Sequences may have several branches executing at the same time. In UML state machines, an analogous feature would be the possibility for a system to be in two (or more) states at the same time. This requires using composite states, each within a region of its own.

Another modeling notation of UML that the ASD notation resembles (both graphically and semantically) is the activity diagram notation that would enable concurrent Sequences of activities and explicit constraints on flows but no timing constraints. However, UML activities cannot be broken up to concepts corresponding to Entry, Step, and Exit Activities of Steps. Activity diagrams may additionally contain decision nodes for which there are no corresponding concepts in ASDs. Lastly, activity diagrams usually describe workflows of entire systems, whereas in UML AP Sequences are used to describe sequential behavior of individual AFs.

Because of the mentioned conceptual differences to the modeling notations of UML that have similar appearance, it was not possible to use directly research work that has been previously done to enable their simulation.

4.2. Model Transformation for Simulating Sequences. In general, Modelica is an equation based language so that the values

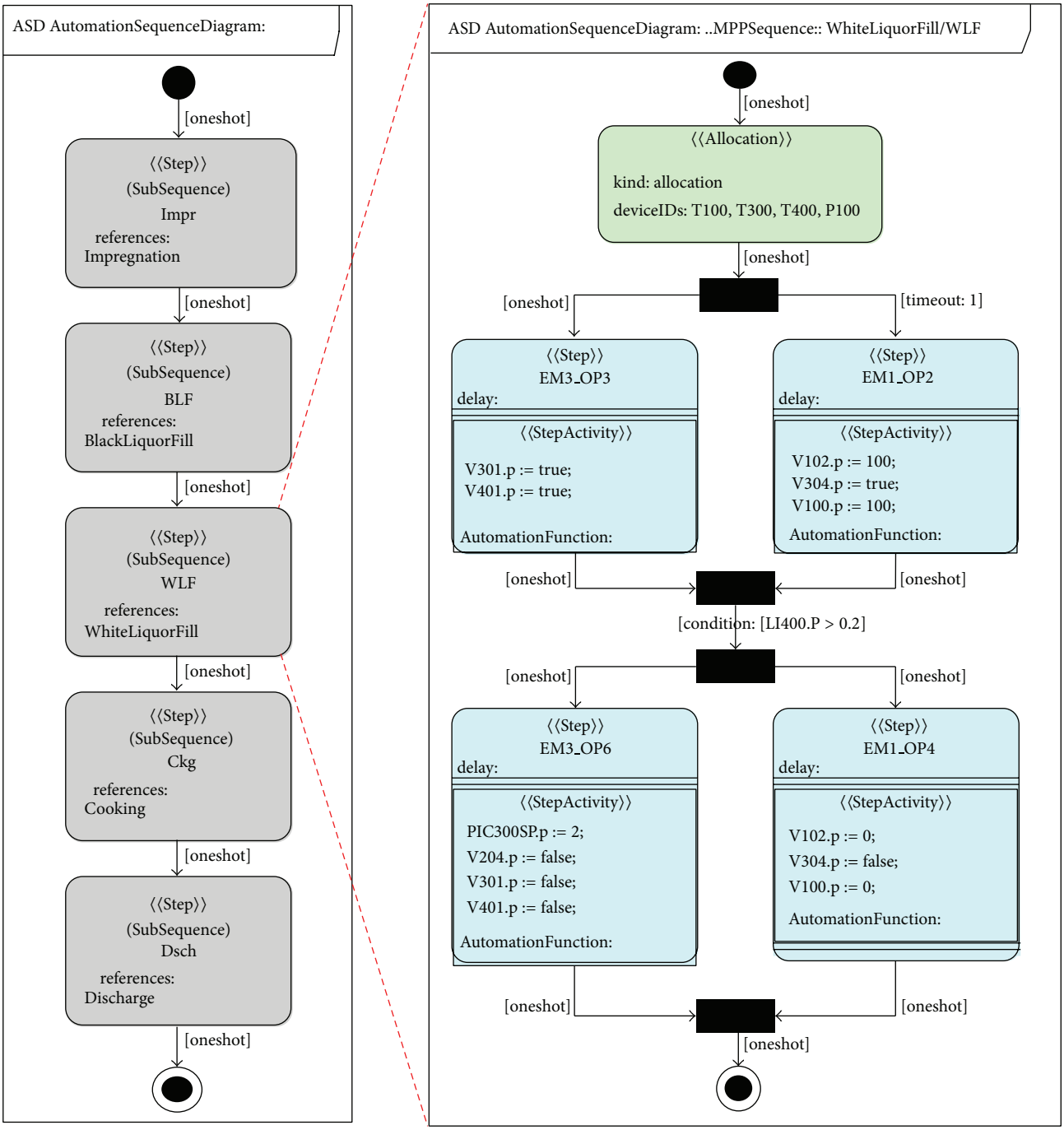

Figure 3: Automation sequence diagrams illustrating a sequence and a referenced subsequence of it.

of variables of Modelica models are determined by equations. However, in addition to the equations that apply all the time, the language includes an algorithmic concept for calculations in which statements are applied in an order. Algorithms are also the constructs of the language that the transformation uses for simulating the Sequences.

The simplified (hiding unnecessary details) metamodel of the ASD diagram type is presented in Figure 4. In the metamodel, Sequence is extended from the UML state machine. The Step and Allocation concepts are extensions of UML state. Entry, Step, and Exit Activities are extended from the UML activity concept and contained by Steps with metamodel properties of UML State (that are hidden from the figure). In addition to the concepts that are shown in the figure, ASDs may contain instances of the mentioned pseudostates of UML, namely, initial, join, and fork states

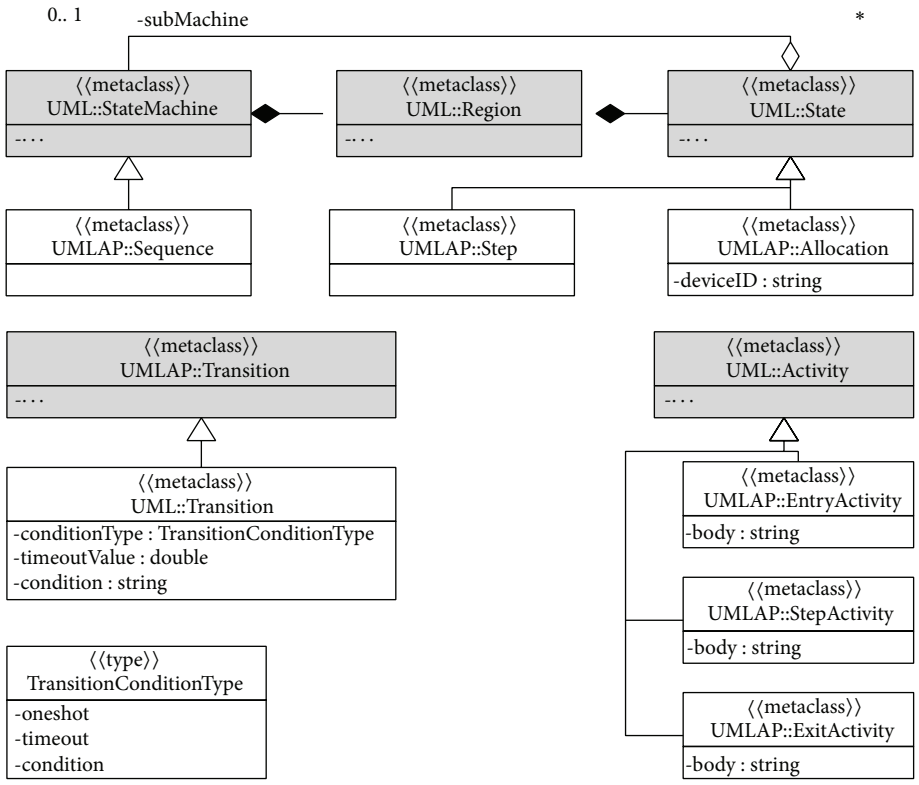

Figure 4: Simplified metamodel of the ASD diagram type with relations to the UML metamodel.

(Steps) as well as final states. Transitions between Steps, Allocations, and pseudo Steps are modelled with a Transition concept that has been extended from UML Transition.

To simulate the behavior of Sequences, they are used as a basis for creating variables and algorithmic code. The systematically named variables are used to keep track of the execution, whereas the algorithmic code changes the values of the variables. The (Entry, Step, and Exit) Activity code of Steps is also included in the algorithms. The variables, which are created to be owned by a Modelica class that corresponds to the AF that owns the Sequence, are created as follows. For the sequence that an ASD represents, and, for each sub-Sequence that is referenced from the Steps of the Sequence, a Boolean variable with the same name than the name of the Sequence is created. These variables are used to indicate the execution of the Sequence being in the Sequence or sub-Sequence in question. In addition, exactly one UML OpaqueBehavior for each highest level Sequence is created to contain the algorithmic code to be generated.

For each Step in a Sequence, the transformation creates two variables. First is a Boolean variable with a name consisting of the name of the Sequence and the name of the Step. The second variable is an Integer variable with a name consisting of the name of the sequence, the name of the Step, and "Phase" literal. The Boolean variables are used to indicate the execution of the Sequence being in the Step in question, whereas the Integer variables keep track of which Activities (Entry, Step, or Exit) have been executed in a Step.

For (exactly one) initial Step in a Sequence, the transformation creates a Boolean variable with a name consisting of the name of the Sequence and "Initialized" literal. For a final Step in a Sequence, the transformation creates a Boolean variable with a name consisting of the name of the Sequence and the name of the final Step. These variables indicate whether or not the execution has reached the initial and final steps in question.

For each fork-to-join region the transformation creates a Boolean variable with a name that consists of the name of the fork, the name of the join, and "Region" literal. In addition, a Boolean variable is created for each branch going out from the fork and coming into the (exactly one) join. The names of these variables consist of the name of the fork (Step) and the number of the branch. The variables corresponding to the branches are used in guard conditions for exiting the join (Step), whereas the other variables are used to indicate the execution of the Sequence being in the fork-to-join region.

For Transitions, the transformation creates variables only if their transition condition is of type timeout. In this case, the name of the real valued variable consists of the name of the Sequence, the name of the Step from which the transition starts, and "Time" literal. These time variables keep track of completion times of the Steps that the Transitions exit from.

Table 1: Mappings between UML AP and UML (ModelicaML) metamodel elements.

Source model (UML AP)	Target model (UML with ModelicaML)		
Element	Model element	Element name	Element type
Sequence	Property	Seq. name	Boolean
	Opaque Behavior	Seq. name + "Algorithm"	—
(UML) Initial (pseudostate)	Property	Seq. name + "Initialized"	Boolean
Step	Property	Seq. name + Step name	Boolean
	Property	Seq. name + Step name + "Phase"	Integer
(UML) FinalState	Property	Seq. name + FinalState name	Boolean
(UML) Fork (pseudostate)	Property	Seq. name + Fork name + "Branch" + #	Boolean
(UML) Join (pseudostate)	Property	Fork name + Join name + "Region"	Boolean
Transition	Property	Seq. name + Step name + "Time"	Double
Allocation	Property	Seq. name + allocation name	Boolean
	Class	"Allocations"	—
	Property	Device ID	Integer

Lastly, for the Allocations, the transformation generates a record (class) and a property for each individual device ID that becomes reserved in the Sequences owned by the AF. The mappings between UML AP and UML metamodel elements are also presented in Table 1.

Some of the algorithmic constructs that are created based on the ASDs are illustrated in an example in Figure 5. First, a when-construct is created that is executed only once at the 7 start of the simulation ("when initial() then"). It sets all the Boolean phase variables (Steps, pseudo Step, Allocations, and sub-Sequences) to false. The Integer variables related to Steps are set to 0 to indicate that no activities have been performed. The Integer variables related to Allocations are also set to 0, to indicate that no allocations are active. The initialization code is created only for each highest level Sequence, not for referenced sub-Sequences.

Steps, Allocations, sub-Sequences, and pseudo Steps are handled with conditional (if-else if) code blocks that can be all entered only once. This is necessary because Modelica models are executed cyclically. In a cycle the execution must continue from the phase to which the execution ended in the previous cycle. For example, arriving to the allocation phase in the example is enabled in the initialization phase and disabled in the allocation phase, which in turn enables the next phase.

Entry, Step, and Exit Activities are executed only once so that when arriving to a Step, the Entry Activity is executed first in addition to changing the phase value to 1. Next, Step Activity is executed and the phase value set to 2. The execution of the Exit Activity and setting the phase variable to 3 waits until the transition condition (if any) to next Step in the Sequence is satisfied so that the transition can occur immediately after performing the Exit Activities. If the Step in question does not contain Entry, Step, or Exit Activities, the corresponding algorithmic code only changes the value of the phase variable.

Allocations are assumed to be next to initial Steps in Sequences. They are intended to model allocations of devices that have IDs corresponding to the ID variables of Allocations. For Allocations, the algorithmic code increases (by one) the variables of the record that correspond to the allocated IDs. At the end of Sequences, allocations are relieved by decreasing the values of the variables by one. In the simulations, the Allocations thus do not force execution to wait but only warn about double allocations, which are indicated by the values of the variables becoming greater than 1. Such problems can then be inspected by developers.

Fork-to-join regions are in the approach handled by creating variables for each branch in the region. The branches may execute independently of each other but for a transition to exit a join Step, all branches must have reached the join. This condition is used as an exit guard for the join, in addition to possible transition conditions related to the transition exiting it.

In the approach, the Modelica code structures resemble the structures in [31] that are used for Modelica simulating state machines. The most notable differences are as follows. Steps or sub-Sequences that are next to another sub-Sequence are not enabled until the sub-Sequence reaches its final Step. This prevents a transition in a higher-level sequence to fire before the final Step is reached. The phases of Steps, that is, whether the Entry, Step, and Exit Activities have been executed, are recorded with Integer variables. The transition conditions to exit Steps are used inside the Steps as guards for shifting to the Exit Activities and enabling the next Step/sub-Sequence. In Allocations that do not have activities the transition conditions are similarly used as conditions to enable the next Steps or sub-Sequences. In referenced sub-Sequences, the next Steps or sub-Sequences are enabled in the final Steps. Lastly, in case of a transition containing a timeout condition, a real valued time variable is created for the previous Step the value of which is set to equal the completion time of the previous Step, pseudo Step, or sub-Sequence. The time variables can be used in the transition conditions as illustrated in Figure 5.

4.3. Constraints and Assumptions. In development of the modeling and simulation approach, a decision was made that Sequences must always be owned by AFs. In this way, the variables and algorithmic code corresponding to a Sequence can be created for a Modelica class corresponding to the AF that owns the Sequence. In the approach a Sequence thus describes the sequential behavior of the AF that owns

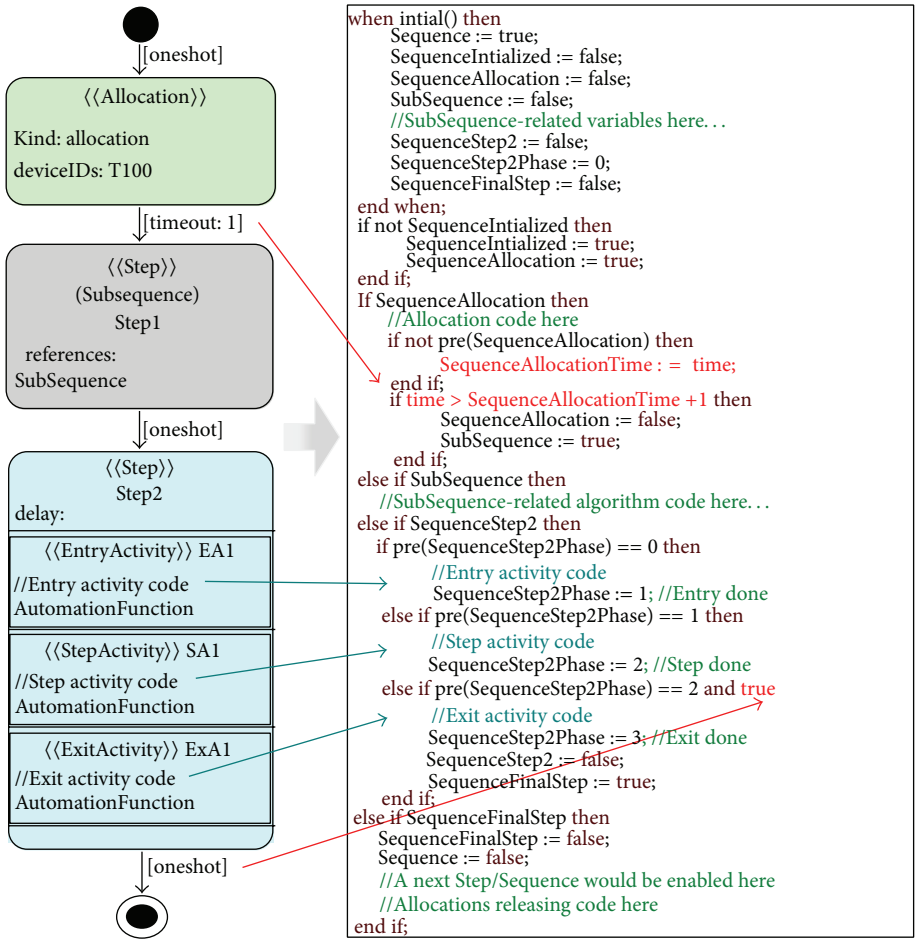

Figure 5: An automation sequence diagram with a corresponding (Modelica) algorithm section.

the Sequence. A Step being executed in a Sequence is a Step of the AF. For control or other signals to be forwarded to other AFs, the AF must be connected to them with use of ports in the interfaces of the AFs. The execution of a single Sequence is thus centralized in an AF. However, an AF may contain several Sequences. On the other hand, a control application model may contain several AFs that define Sequences so that at runtime there would be several Sequences executing concurrently and independently of each other.

The properties that are created for implementing the sequential behavior (see previous section) become the properties of the (ModelicaML) class that is created to correspond to the AF. The properties are necessary for implementing the dynamic behavior, by controlling the execution of algorithmic statements. During simulation, however, they also indicate the execution of the Sequence. For example, the Boolean valued properties created to correspond to Sequences (and its possible sub-Sequences) have value true only when the execution is in the sub-Sequence in question. This feature has been used, for example, in Figure 9 in which the upper plot presents the sub-Sequences of a pulp batch processing Sequence.

There are also restrictions related to the use of Sequences in the approach. Currently, for the simulation transformation to work properly, fork-to-join regions must be balanced so that branches exiting a fork meet each other in one join. On the other hand, the transformation does not support loops within Sequences so that a Step could be entered more than once in a Sequence. It is also assumed that a Sequence always contains an initial Step and at least one final Step. Whether the restrictions related to initial and final Steps hold is checked before performing the transformation.

4.4. Implementation of the Approach. The transformation for simulating Sequences was implemented by extending the previous version of the transformation [1]. In addition to Control Structure and logic diagrams that were supported by the previous version, the transformation processes Sequences contained by AFs to properties and algorithm sections of Modelica classes. The core of the transformation was written with the QVT (Query/View/Transformation) operational mappings language [29] and the executable Java code generated with the SmartQVT tool. The generated Java transformation class was complemented by extending it with

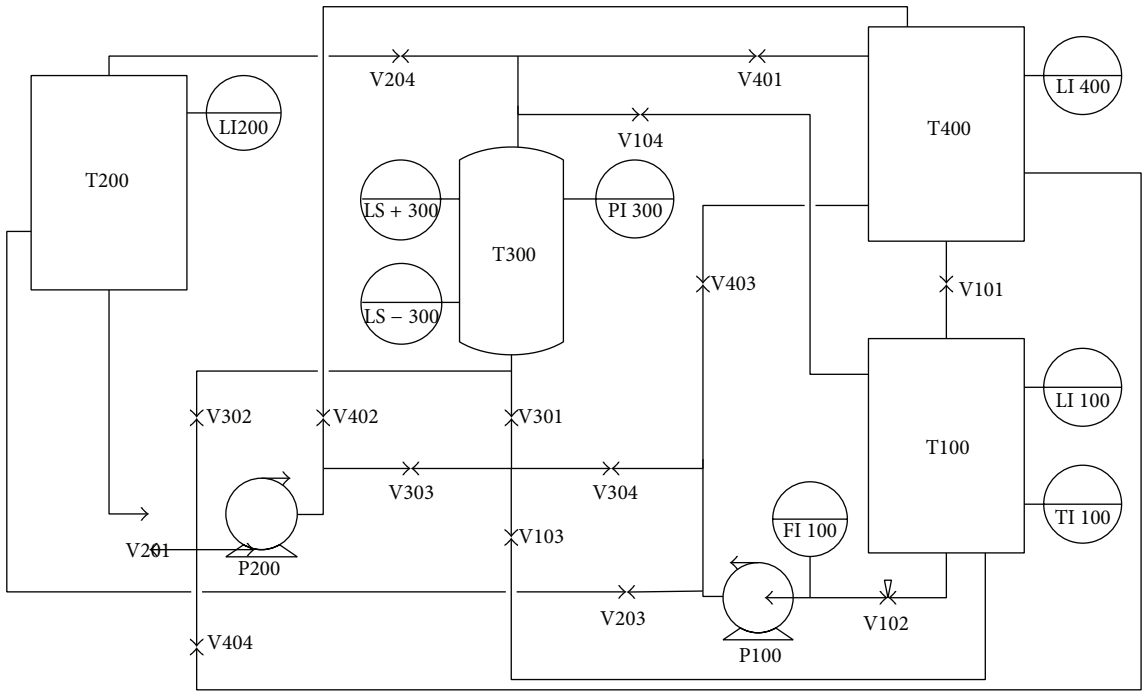

Figure 6: P&I diagram of the pulp production process.

a manually written class. It takes care of, for example, the creation of tagged values of stereotypes and other tasks that are hard to express with QVT languages. To enable launching the transformation with the graphical user interface of the supporting UML AP tool [28], the transformation was packaged to an Eclipse plugin. The plugin architecture and integration to the tool was implemented as outlined in [32].

5. Illustrative Example

The example system to be used in this paper is a laboratory scale pulp processing plant the piping and instrumentation (P&I) diagram of which is in Figure 6. The plant includes 3 storage tanks, a boiler, 2 pumps, 2 control valves, 13 solenoid valves, and piping that enable pumping fluid from any tank to the boiler and via boiler back to any of the tanks. The tanks contain instrumentation to measure liquid levels in them, temperature in tank T100, and pressure in boiler T300. The process is used to simulate batch processing of pulp which is located in the boiler and processed with process substances (impregnation liquor, black liquor, and white liquor) according to timing, pressure, and temperature constraints. In specific phases of the processing sequence, feedback control is required to control the temperature of the white liquor (in tank T100) and pressure in the boiler (T300).

To enable the simulation of the process with a modelled control solution, the process was modeled with ModelicaML. This included defining simulation classes for the physical parts of the process including tanks, boiler, pumps, solenoid valves, control valves, pipes, and pipe crossings with 3 and 4 inlets. Tanks keep record on liquid levels and temperatures inside them. For temperature equations, ideal mixing of fluids is assumed. The liquid flows in pipes and in control valves that are proportional to constants measured from the process and to square roots of the pressure differences between the ends of the pipes/valves. Pumps increase the pressure in their output sides and solenoid valves stop the liquid flows regardless of the pressure differences.

The simulatable ModelicaML model was then defined by creating instances of the classes and connecting them together according to the connections in the physical process. This was done with a structured class diagram. A small part of the diagram, related to the surroundings of the tank T400, is presented in Figure 7.

The control solution for the batch process is illustrated with Figures 8 and 3. Figure 8 presents a (UML AP) Control Structure Diagram of the control solution. It contains binary and analogue valued input and output AFs for interfacing with the sensors and actuators of the process. The Sequence is implemented within the MPPSequence AF that controls some actuators directly and uses controllers for controlling T300 pressure (by throttling valve V104) and T100 temperature with heater E100. To illustrate how logic diagrams and ASDs are used to define behavior of AFs, the figure has been complemented with the MPP Sequence and a logic diagram definition of the temperature controller.

The other illustrating figure (Figure 3) was used as an example earlier and illustrates the MPPSequence and one of the sub-Sequences of it, WhiteLiquorFill. MPPSequence consists of 5 main phases: Impregnation, BlackLiquorFill, WhiteLiquorFill, Cooking, and Discharge. During the phases, the boiler is filled with impregnation liquor and pressurized, filled with black liquor that replaces the impregnation liquor (BlackLiquorFill), filled with white liquor that replaces

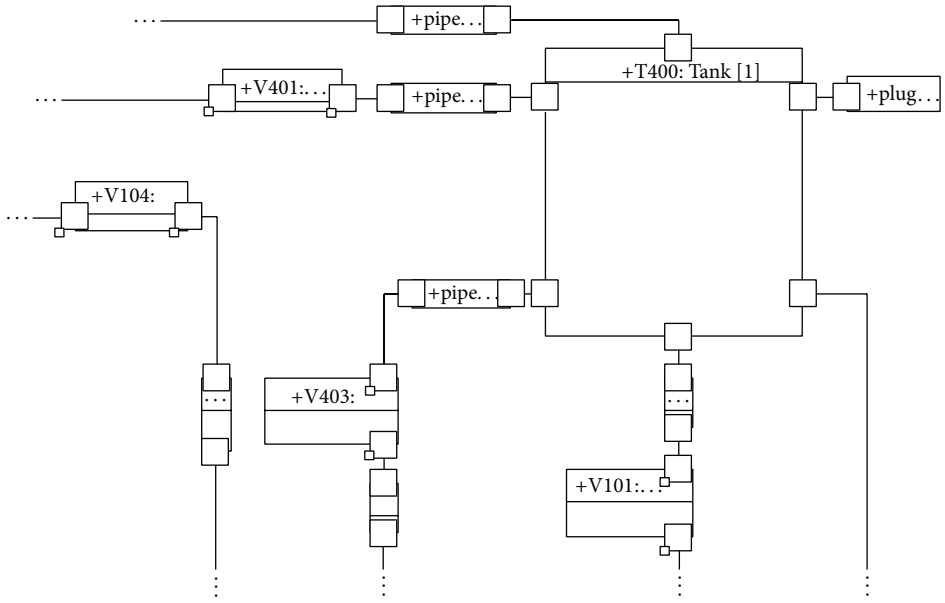

Figure 7: A part of the ModelicaML plant model.

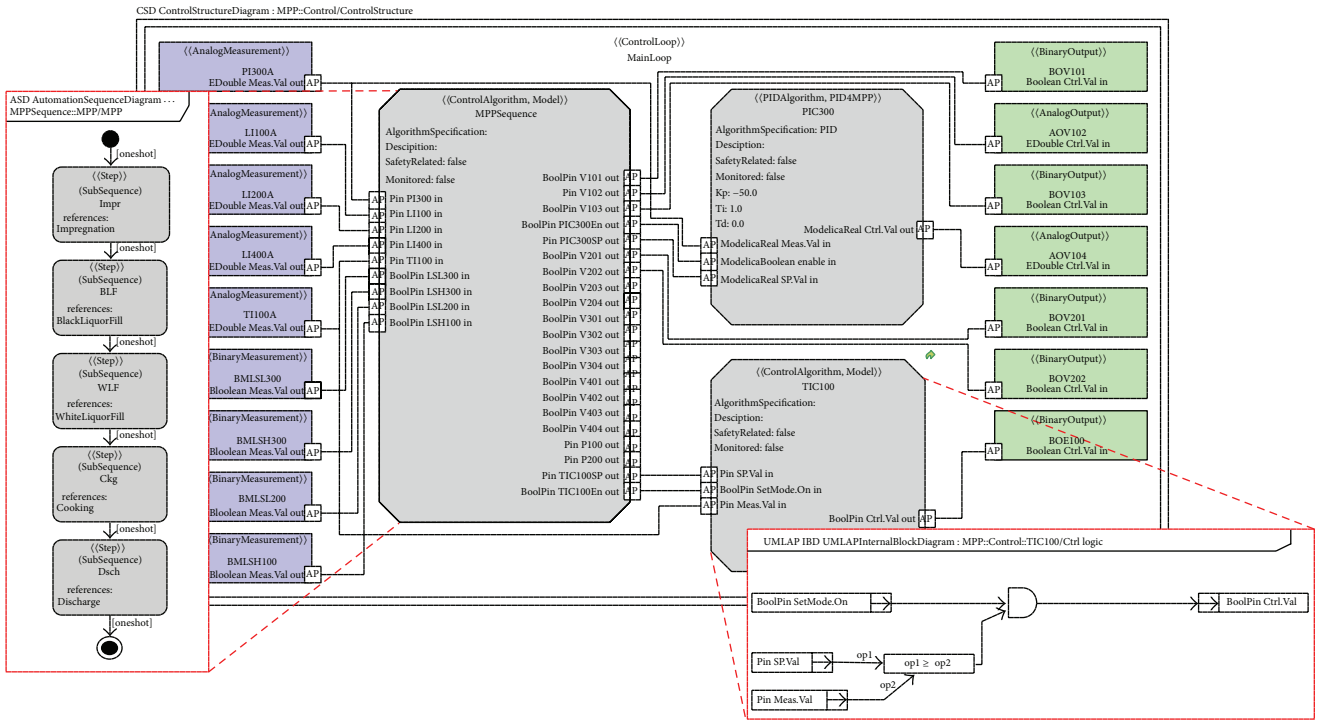

Figure 8: Modeling of automation sequences integrates to previous work with control structure and logic diagrams.

the black liquor (WhiteLiquorFill), heated to cooking temperature and pressurized (Cooking), and finally drained back to white liquor tank T400 (Discharge). WhiteLiquorFill, on the other hand, opens valves V301, V401, V102, and V304, and pumps liquor until the level in tank T400 exceeds 0.2 (m).

In order to obtain simulation results of a closed-loop system, the developed transformation was used to transform and connect the control system model to the plant model. Practically this included selecting the simulator export functionality of the tool and the (target) ModelicaML plant model file. After performing the transformation, the model was simulated with OpenModelica [26] tools. The ModelicaML model was first transformed to Modelica code and then loaded to the simulator environment. Initial values for

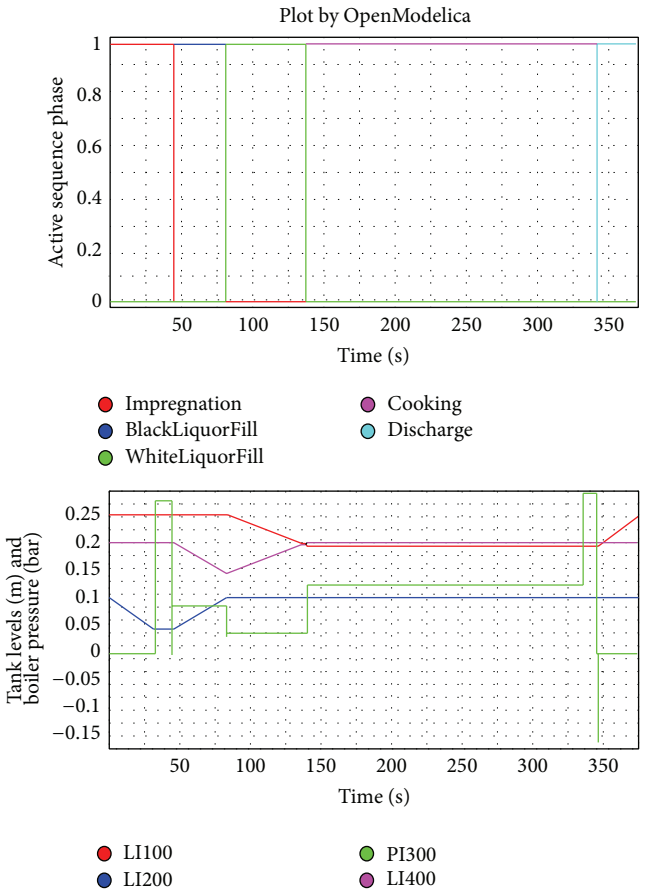

Figure 9: Simulation results plotting active phases of the sequence, levels in tank T100 (LI100), T200 (LI200), and T400 (LI400) as well as pressure in boiler T300 (PI300).

the plant, for example, levels and temperatures in the tanks, were defined in the process model. For different simulations they could have been changed at this point too.

A plot illustrating the results from simulating the Sequence is shown in Figure 9. The main phases are plotted in the upper part of the figure, a value being one indicating execution of the phase in question. The lower part of the figure plots the levels of liquor in tanks T100 (LI100), T200 (LI200), and T400 (LI400) and pressure in tank T300 (PI300). According to the results, the control solution including the Sequence works as intended. Processing liquors are used in the correct order and the boiler pressurize, during black liquor fill and cooking phases.

The values shown in the figure were selected for plotting after performing the simulation. The simulator keeps record on all variables related to a simulation. Any other set of variables related to an aspect in the process or in the control solution, for example, functioning of a controller, could have been selected for plotting as well.

6. Discussion

This paper has addressed the issue of simulating sequential control activities within MDD of control applications. The approach integrates to the previous work of the authors and enables the use of Automation Sequence Diagrams (ADSs) of UML AP to define sequential behavior of Automation Functions for simulation purposes. The transformation to simulatable ModelicaML form was implemented with open source modeling and model transformation (QVT) tools on the Eclipse platform. The ASD diagram type that is in the approach used for modeling sequential control has been extended from UML state machine diagrams. However, because of significant differences in execution semantics of state machines, it was not possible to rely on existing work [31] related to simulating them in Modelica form.

The benefits from using Modelica (ML) as the (target) simulation language of the approach included the ability to use standard model transformation techniques. Modelica is also an object-oriented simulation language, which was taken the advantage of mainly in development of the plant simulation model. From the point of view of simulating the control application, however, object-oriented features were not used. As a consequence, it is expected that the presented approach could be used also with other simulation languages that can be accessed with model transformations, for example, Simulink. An approach to execute Sequences without equation based, acausal execution semantics of Modelica could also be similar to the one presented in this paper. Algorithmic constructs were used also in case of Modelica instead of equations that apply all the time.

The novelty of the simulation approach is in the ability to simulate control application models at design time, before IEC language [17, 18] implementations of the applications. Closed-loop MiL simulations are created with model transformations so that a genuine simulation language (Modelica) is used for simulating both plant and control application models. Other MDD approaches in the domain (in which simulations have been supported) have utilized IEC 61499 as a simulation (in addition to implementation) language [15, 16, 20] or relied on the use of cosimulation [14, 19]. On the other hand, sequential control as a special aspect of control systems has been addressed only in [13] but not with respect to simulations. With the work presented in this paper and [1], the simulation approach covers all the common aspects of basic control systems including binary and feedback control, sequential control, and interlocks.

An issue that is not yet addressed in the approach [1] is delays in control systems hardware, for example, networks in distributed control systems. However, the objective of the approach is to enable simulations early, already before, for example, finishing control system hardware design. On the other hand, effects of delays and, for example, random noise in instrumentation can be included in the models, in simulation classes of sensors and actuators of the process. It is also assumed that, for example, delays in typical control system hardware are less significant than those in instrumentation.

Support for model-based control software development is also part of some commercial products. For example, B&R (Automation Studio) [33] and Beckhoff (TwinCAT 3) [34] support the development of control applications in MATLAB/Simulink environment and generating executable (PLC) code based on the models. As a difference to such

products, the work presented in this paper intends to support simulations in an MDD approach in which all models are not simulatable. Instead, models are developed gradually from requirements towards executable applications using model transformations for shifting between models and, for example, importing source information to models. In addition, the models cover special needs such as traceability between requirements and design artifacts that are becoming more and more important in the domain.

To illustrate the simulation approach, it was applied to simulation of a controlled pulp batch production process. For the case study, the pulp production process was modelled with Modelica. Flow, pressure, and temperature equations for all the plant components in the model led to the total number of equations for the closed-loop system to be approximately 1400. As such, the closed-loop system was the largest that has been utilized in the simulation experiments of the approach so far. It also demonstrates the scalability of the approach for practical, nontrivial simulation needs.

7. Conclusions

MDD techniques are under active research in the application domain of industrial control systems. However, despite the research activities, and the tradition of using simulations, simulations have not yet been sufficiently integrated to MDD in the domain.

In MDD, it is possible to utilize model transformations for obtaining simulation models already before programmed implementations of the applications. This possibility should be taken advantage of. Control applications models should be evaluated in a timely manner and in closed-loops with the models of the processes to be controlled. In order to relieve control application developers from the task of coupling simulation engines, the simulations should follow the model-in-the-loop approach using a single simulation engine.

The presented approach complements the simulation approach of the authors with the possibility to simulate sequential control activities in conjunction to feedback and binary control as well as interlocks. The new work has been targeted for the sequences of process and batch industry. However, control sequences can be beneficial also in simulations of other kinds of processes. For example, in a previous simulation experiment [1], the set-point trajectories to evaluate a control system in different conditions needed to be defined manually. With the work presented, the set-point trajectories can be included in Sequences of the models.

According to our experiences, the simulation approach is useful in revealing defects in control algorithms, structures, and tunings. The simulations can be performed already at design time and so that decisions made in a development phase can be evaluated before they affect decisions in later phases. By creating simulation models with automated model transformations, simulations can be used as a continuous, design-time quality assurance method. This can be done without causing excessive additional workload to developers.

It is also expected that the task of developing models of the processes to be controlled with Modelica becomes easier and more attractive for industry in near future. This is due to improvements in libraries of simulation classes, the Modelica standard library, from which it is possible to compose plant models. It is also a clear benefit of Modelica that it includes support for standard and user/company specific libraries. Modelica is already supported by both commercial and open source tools that can be used by both industry and academy.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] T. Vepsäläinen and S. Kuikka, "Simulation-based development of safety related interlocks," in *Simulation and Modeling Methodologies, Technologies and Applications*, pp. 165–182, Springer, 2013.

[2] D. Hästbacka, T. Vepsäläinen, and S. Kuikka, "Model-driven development of industrial process control applications," *Journal of Systems and Software*, vol. 84, pp. 1100–1113, 2011.

[3] T. Ritala and S. Kuikka, "UML automation profile: enhancing the efficiency of software development in the automation industry," in *Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN '07)*, pp. 885–890, June 2007.

[4] H. Shokry and M. Hinchey, "Model-based verification of embedded software," *Computer*, vol. 42, no. 4, pp. 53–59, 2009.

[5] A. Plummer, "Model-in-the-loop testing," *Proceedings of the Institution of Mechanical Engineers I*, vol. 220, pp. 183–199, 2006.

[6] H. Chae, X. Jin, S. Lee, and J. Cho, "TEST: testing environment for embedded systems based on TTCN-3 in SILS," *Communications in Computer and Information Science*, vol. 59, pp. 204–212, 2009.

[7] M. Short and M. J. Pont, "Assessment of high-integrity embedded automotive control systems using hardware in the loop simulation," *Journal of Systems and Software*, vol. 81, no. 7, pp. 1163–1183, 2008.

[8] M. Schlager, R. Obermaisser, and W. Elmenreich, "A framework for hardware-in-the-loop testing of an integrated architecture," in *Software Technologies for Embedded and Ubiquitous Systems*, pp. 159–170, Springer, 2007.

[9] G. Stoeppler, T. Menzel, and S. Douglas, "Hardware-in-the-loop simulation of machine tools and manufacturing systems," *IEE Computing and Control Engineering*, vol. 16, no. 1, pp. 10–15, 2005.

[10] J. A. Carrasco and S. Dormido, "Analysis of the use of industrial control systems in simulators: state of the art and basic guidelines," *ISA Transactions*, vol. 45, no. 2, pp. 295–312, 2006.

[11] MODELISAR Consortium, "Functional Mock-up Interface for Co-simulation," Version 1.0., 2010.

[12] G. Hemingway, H. Neema, H. Nine, J. Sztipanovits, and G. Karsai, "Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach," *Simulation*, vol. 88, no. 2, pp. 217–232, 2012.

[13] T. Lukman, G. Godena, J. Gray, M. Heričko, and S. Strmčnik, "Model-driven engineering of process control software-beyond device-centric abstractions," *Control Engineering Practice*, vol. 21, no. 8, pp. 1078–1096, 2013.

[14] H. Thompson, D. Ramos-Hernandez, J. Fu, L. Jiang, J. Nu, and D. Dobinson, "The FLEXICON co-simulation tools applied to a marine application," *Proceedings of the Institution of Mechanical Engineers M: Journal of Engineering for the Maritime Environment*, vol. 222, pp. 81–94, 2008.

[15] V. Vyatkin, H. Hanisch, C. Pang, and C. Yang, "Closed-loop modeling in future automation system engineering and validation," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 39, no. 1, pp. 17–28, 2009.

[16] I. Hegny, M. Wenger, and A. Zoitl, "IEC 61499 based simulation framework for model-driven production systems development," in *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '10)*, September 2010.

[17] International Electrotechnical Commission, *IEC 61499-1: Function Blocks-Part 1: Architecture*, International Standard, Geneva, Switzerland, 1st edition, 2012.

[18] International Electrotechnical Commission, *IEC 61131-3: Programmable Controllers part 3, Programming Languages*, IEC Publication, 2013.

[19] L. Ferrarini and A. Dedè, "A model-based approach for mixed Hardware in the Loop simulation of manufacturing systems," in *Proceedings of the 10th IFAC Workshop on Intelligent Manufacturing Systems (IMS '10)*, pp. 36–41, July 2010.

[20] C. Yang and V. Vyatkin, "Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems," *Control Engineering Practice*, vol. 20, no. 12, pp. 1259–1269, 2012.

[21] J. Provost, J. Roussel, and J. Faure, "Translating Grafcet specifications into Mealy machines for conformance test purposes," *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, 2011.

[22] R. David, "Grafcet: a powerful tool for specification of logic controllers," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 3, pp. 253–268, 1995.

[23] A. Hellgren, M. Fabian, and B. Lennartson, "On the execution of sequential function charts," *Control Engineering Practice*, vol. 13, no. 10, pp. 1283–1293, 2005.

[24] P. Fritzson and V. Engelson, "Modelica—a unified object-oriented language for system modeling and simulation," in *Proceedings of the Object-Oriented Programming Conference (ECOOP '98)*, pp. 67–90, Springer, 1998.

[25] W. Schamai, *Modelica Modeling Language (ModelicaML): A UML Profile for Modelica*, Linköping University Electronic Press, 2009.

[26] OpenModelica, https://www.openmodelica.org/.

[27] Dassault Systemes, Dymola, http://www.3ds.com/products-services/catia/capabilities/systems-engineering/modelica-systems-simulation/dymola.

[28] T. Vepsäläinen, D. Hästbacka, and S. Kuikka,, "Tool support for the UML automation profile—for domain-specific software development in manufacturing," in *Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA '08)*, pp. 43–50, 2008.

[29] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (QVT), Version 1.0," Object Management Group, 2008.

[30] T. Vepsäläinen and S. Kuikka, "Benefit from simulating early in MDE of industrial control," in *Proceedings of the IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA '13)*, pp. 1–8, 2013.

[31] W. Schamai, U. Pohlmann, P. Fritzson, C. J. Paredis, P. Helle, and C. Strobel, "Execution of UML State machines using modelica," in *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT '10)*, pp. 1–10, 2010.

[32] T. Vepsäläinen, D. Hästbacka, and S. Kuikka, "A model-driven tool environment for automation and control application development-transformation assisted, extendable approach," in *Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, pp. 315–329, 2009.

[33] B&R Automation Studio, http://www.br-automation.com/en/products/software/automation-studio/.

[34] Beckhoff TwinCAT 3, http://www.beckhoff.fi/english.asp?twincat/default.htm.

Publication 5

Vepsäläinen, T., Kuikka, S. (2013) Benefit From Simulating Early in MDE of Industrial Control. Proceeding of the 18th IEEE International Conference on Emerging Technologies and Factory Automation. Cagliari, Italy, September 10-13, 2013, pp. 1-8.

DOI: 10.1109/ETFA.2013.6647961

Benefit from Simulating Early in MDE of Industrial Control

Timo Vepsäläinen
Tampere University of Technology,
Dept. of Automation Science and Engineering
Korkeakoulunkatu 3, 33101, Finland
timo.vepsalainen@tut.fi

Seppo Kuikka
Tampere University of Technology
Dept. of Automation Science and Engineering
Korkeakoulunkatu 3, 33101, Finland
seppo.kuikka@tut.fi

Abstract

This article focuses on integration of simulations to model-driven engineering (MDE) of automation and control systems and applications. MDE offers means to automate repetitive design tasks and thus improves the efficiency of development work. However, it does not reduce the need for genuine decisions of professional developers to challenging design tasks. Formerly, the decision making has been facilitated with separate process simulations to predict the characteristics of controlled processes. This paper presents and evaluates a new way to seamlessly integrate simulations to early phases in MDE of control applications. On one hand, we argue why and how simulations should be organized in MDE in the domain. On the other hand, we present and summarize observations from the experiments in which our simulation approach has been used.

1. Introduction

This article focuses on integration of simulations to model-driven engineering (MDE) of automation and control applications. MDE is a system and software development methodology that emphasizes the use of models and model transformations instead of, for example, textual documents. In MDE, models conform to formally specified modeling languages such as UML (Unified Modeling Language) and are processed with model transformations. Models contain the design information for both development and documentation purposes. Model transformations, on the other hand, enable processing of models to create and update models as well as code implementations.

Model transformations may automate importing information to models from models of preceding development phases, e.g. process design, and their tools. Design models can be used for creating analyzable models that can be studied using application domain specific tools. Automated model checks may reveal problems and inconsistencies in models and between modeled phase products of the development process. The mentioned benefits of MDE are mainly related to automating repetitive tasks that are time-consuming but possible to handle with pre-defined rules that can be programmed to model checks or transformations.

However, the use of MDE has not been able to automate all the complex design and development tasks. Demanding design tasks and decisions over alternative solutions to achieve (sometimes informally) specified objectives and system characteristics need to be made by professional developers. Nevertheless, developers do not always have to rely solely on their experience on design tasks. Simulations have enabled comparing alternative solutions and predicting the characteristics of systems and controlled processes based on models and partially implemented systems in conjunction to models.

Despite the research activities to utilize MDE techniques - and the long tradition of using, e.g. Matlab/Simulink based techniques - simulations have not yet been sufficiently integrated to MDE in the domain. Because models can be used for automating generation of code, it should be also possible to generate simulation models to be used with simulation models of the processes to be controlled. In this way, MDE could provide even more benefits compared to traditional control application development approaches. By applying model transformations to generation of simulation models, simulations could be made a part of continuous development-time quality assurance work.

In our previous work, we have developed simulation integration [1] to our tool-supported MDE approach [2] for control applications of both process industry and machinery systems. The integration covers modeling and simulation of cyclically and also sequentially [3] executed control functions that can be librarized for later use and to support use of vendor specific libraries. The modeling in the approach is based on UML Automation Profile (UML AP) for which we have developed tool support on Eclipse platform. The integration is based on Modelica and model transformations for creating ModelicaML (Modelica Modeling Language) simulation models. An example industrial process from our latest study is presented in figure 1.

The contributions of this article are to assess the benefits of integrating simulations to MDE in the automation domain and to evaluate and compare our simulation approach to other approaches. Related to using simulations in general, we argue why MDE in the

automation domain should focus on Model-in-the-Loop (MiL) simulations. We point out the benefits of early MiL simulations in comparison to simulation approaches enabled by control system platforms and compare alternative approaches to implement MiL simulations. Related to our approach specifically, we present lessons learned and observations from several simulation experiments in which we have modeled control systems for both industrial processes and machinery systems.

The rest of this paper is organized as follows. In section 2, we present work related to integrating simulations to MDE and using simulations in MDE of automation and control systems. In section 3, we argue why MDE in the domain should focus on MiL simulations, compare early MiL simulations to other possible simulation approaches, and compare alternative approaches to implement closed-loop MiL simulations. Before concluding the paper, in section 4 we briefly present our tool-supported simulation approach and the observed benefits of it in three simulation experiments.

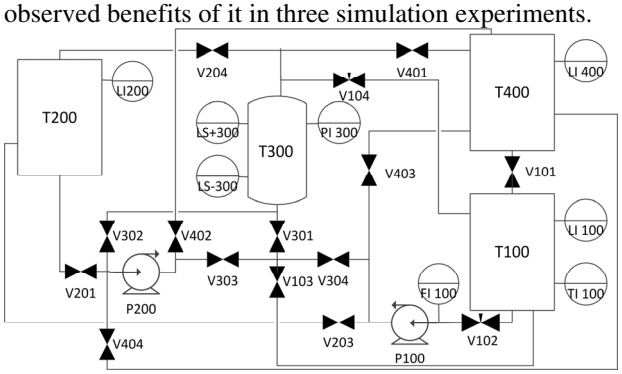

Figure 1. The pulp batch production process used as an example process in [3]

2. Related work

General simulation approaches that can be utilized in conjunction to MDE include model-in-the-loop (MiL), software-in-the-loop (SiL), processor-in-the-loop (PiL) and hardware-in-the-loop (HiL) simulations [4]. These approaches are in [4] addressed in development of embedded systems. In MDE of automation and control systems and applications the differences between the approaches are in the control system configurations used to control the plant simulation models. For example, in MiL a model of the control system/application is used whereas SiL, PiL and HiL utilize generated software, generated software with target processor and generated software with entire target hardware, respectively. Accordingly, they also differ in the nature of defects that they are capable to reveal. For example, MiL simulations evaluate the conceptual control solutions and cannot reveal problems related to software-hardware integration whereas HiL simulations (with the target hardware) can.

Similar simulation approaches, except MiL, can also be utilized in conventional control system development. For example, HiL simulation can be used to test a control application with its target hardware regardless of the development process of the software application. Support for the simulation approaches is also nowadays provided by major proprietary DCS (Distributed Control System) vendors as presented in [5].

Another classification of simulation approaches is related to the amount of simulation engines. Simulation of a controlled system, i.e. a system including a process to be controlled and a control system can be performed within a single simulation engine or as a co-operative simulation (co-simulation). In co-simulation, parts of the overall system are simulated within (2 or more) different but connected environments. The approach, however, requires a mechanism for coupling the simulation environments and replicating commands of them. See [5] for a list of basic simulation functions.

Integration of simulations to MDE of control software has been common in the application domains of embedded and automotive software. For example, [6] describes a general framework for, and two examples of use of MiL simulation. In [7] a testing environment for embedded systems is presented which utilizes TTCN-3 notation in test specifications and SiL simulation for executing the tests. HiL simulation and testing have been utilized for example by Short and Pont [8], Schlager et al. [9] and Stoeppler et al. [10].

In the domain of industrial control, integrating simulations to MDE approaches has not been a principal goal. However, such work has been presented at least in [11], [12] and [13]. In addition, in [14] Ferrarini and Dede have presented a co-simulation approach in which the aim is to test already implemented parts of control systems and applications while simulating the rest. The approach uses HiL simulation but does not confine to MDE techniques for acquiring simulation models. Instead, it is targeted to provide support for flexible co-use of already programmed and simulated control functions with a plant simulation model.

Yang and Vyatkin in [11] utilize MDE techniques the aim being to create IEC 61499 Function Blocks (FB) models from Simulink models in order to verify control applications which are also developed with IEC 61499. Hegny et al. in [13], similarly, create IEC 61499 plant models which are described either with timed state charts (conforming to a project specific metamodel) or external behavior descriptions, which make the approach a co-simulation approach. In [12] the approach of Vyatkin et al. is based on composing applications from intelligent mechatronic components that should include models and with model transformations enable development-time verification and simulation of the applications.

3. Comparison of simulation approaches

The purpose of this section is to discuss, compare and evaluate at a conceptual level the possible simulation approaches in conjunction to MDE of control systems

and applications. In the discussion, we take into account their special characteristics. Some of the characteristics, in addition to arguing why MDE in the domain should concentrate on MiL simulations, will be discussed first in section 3.1. In section 3.2, we discuss the applicability of MiL simulations over target platform specific simulations. In section 3.3 we compare the alternative approaches to implement MiL simulations including their benefits, disadvantages and restrictions.

3.1. Simulations in MDE of control applications

The development of control systems and applications has several characteristics that also affect the MDE of them and how the development process could utilize simulations. To our work, the most important are:

- Many industrial vendors of control system platforms already support connecting the control systems to simulators [5] to support PiL and HiL simulations, depending on whether the connections to plant simulators are implemented directly or via I/O units. Similarly, it is common for industrial DCS vendors to provide support for computer-execution of the control programs to enable SiL simulations. For PLC-based (Programmable Logic Controller) control systems there are soft PLC solutions that enable running programs on desktop computers.
- Industrial control system development utilizes, to the extent possible, librarized blocks for interfacing with the instrumentation of the processes and implementing control algorithms. In some companies in the domain, even metrics on use of librarized blocks have been used to characterize the composed applications. [15].

As a consequence of the characteristics, we argue the following. 1) If MDE techniques are used for developing control applications to be used in PLC or DCS platforms, developing support for other types of simulations than MiL might not be able to provide significant benefits since the other simulation types are already enabled by the platform vendors and usable after generating code. The focus should, thus, be in MiL simulations. 2) Since applications are composed of re-usable library blocks, simulation counterparts of the blocks could be librarized as well and used in MiL simulations. Re-using well-tested, parameterizable simulation blocks could support the confidence in the results of MiL simulations but also simplify the generation of models. Simulation model generation should, thus, be capable of re-using libraries of either simulation or implementation blocks.

However, there are several approaches to perform MiL simulations in MDE. A closed-loop simulation can be a co-simulation in which the model of a control system/application and that of the system to be controlled (plant model) are simulated in different but connected environments. Alternatively, the simulation can be performed within a single simulation engine. This, however, usually requires availability of both the control system and plant models in same simulation language. In practice, this would require:

- control system model and plant model being developed with a (same) language that can be simulated or
- plant model being transformed to the modeling language used to develop the control system model which must be possible to simulate or
- control system model being transformed to the modeling language used to develop the plant model which must be possible to simulate or
- both the control system and plant models to be transformed to a simulation language.

Of these approaches, the last one is arguably the most laborious one and prone to errors since it requires developing and keeping up-to-date two possibly complex model transformations - like in implementing both the second and third alternatives. On the other hand, related to the first approach, the authors are not aware of a language that would support well both the development of simulation models of complex industrial processes and software applications. Additionally, it would need to integrate well with other functional and non-functional information required in MDE of control applications.

Consequently, practical alternatives to enable closed-loop MiL simulations of controlled systems in MDE in the domain are to *use co-simulation* or *transform* either the *plant model* to the language used in control system model or the *control system model* to the language used by the plant model. Of the approaches presented in the related work section, [11] and [13] have chosen the approach to transform plant models. In [14] the approach utilizes co-simulation but with hardware included (HiL). The approach to transform control system models is - according to the knowledge of the authors - utilized only in our work, in the automation domain. The first and last approaches in the bullet list above have not been used.

3.2. Comparing MiL simulations to target platform specific simulations

Use of simulations in automation and control system development, in general, is not a new idea. Simulation solutions are provided by commercial DCS vendors [5] and for PLC based control systems, e.g. soft PLC solutions enable execution of control programs on desktop computers. Benefits of applying simulations have also been discussed in several articles.

In [16] the author compared I/O simulation approach to the traditional approach of performing control system testing only on-site with the actual controlled processes. According to the article, the use of simulations results in shorter start-up times as well as less waste of end products during the start-ups. Use of simulations enables better operator training, ability to test control programs

in smaller modules, and the ability to thorough testing of emergency and dangerous situations.

In his doctoral thesis [17], Karhela mentioned the use of simulations to control system testing, operator training, plant operation optimization, process reliability and safety studies, improving processes, verifying control schemes and strategies, and start-up and shutdown analyses. According to [5], the benefits of using control systems in simulators before installation include improvements to 1) design, development and validation of the control programs and strategies, 2) design, development and validation of the HMI (Human-Machine Interface) and 3) adjustments of control loops and programs. Simulations have thus benefitted control system and application development even before applying model-driven engineering techniques.

In the referred approaches utilizing simulation within MDE, see section 2, the number of anticipated benefits of simulations is smaller. However, their authors can be assumed to have compared their approaches and the benefits implicitly to simulations enabled by the target platforms. Nevertheless, in [13] the objective is to enable *early control application development*, when the plant or equipment is not physically built-up, as well as detection of *inconsistencies* and *missing requirements*.

In [11], the approach is motivated by the ability to provide necessary plant models to be used in simulations to *validate* distributed systems compliant with IEC 61499. Additionally, the article mentions easy re-use of Simulink blocks as IEC 61499 function blocks, potential to improve performance with distributed computation and reducing conflicts between the natures of the two models (IEC 61499 and Simulink). However, the latter advantages are related to use of IEC 61499, not to use of simulations. In [12] simulations are seen to enable *prototyping* and *verification* of applications, although formal verification methods are seen necessary to complement simulation techniques.

To summarize the claimed benefits, it is clear that many of them could be achieved also with simulations enabled by control system platform vendors but not as early. However, the other way round, there should be no reason why early MiL simulations could not be used for simulation tasks that do not require control hardware. For example, verification and validation of control strategies, schemes and tunings, prototyping, and testing in small modules should be possible. But because MiL simulations do not require either the physical plant or control system hardware design to be finished, it is clear that they can be performed earlier.

The inexistence of control hardware in the simulations is also related to another potential benefit. Since the hardware is not needed, simulations could be applied also in companies performing out-sourced design tasks, which would be a significant benefit in networked business environments. Lastly, it should be noticed that using MiL simulation early in the design process does not restrict the use of other simulation approaches i.e. SiL, PiL and HiL later in the process. By selecting a suitable plant modeling language for MiL simulation, it is also possible that the plant model could be re-used in simulations enabled by the control system platform. Our own observations from applying transformation-assisted MiL simulations will be presented later in section 4.

3.3. Comparison of MiL simulation approaches

A clear conclusion of the earlier section 3.1 was that if simulations are integrated to MDE of control applications they should follow the MiL simulation approach. However, as discussed, there are at least 3 practical approaches for obtaining MiL simulations, which have been used in the domain. The approaches are next compared taking into account the restrictions that they place on the modeling languages to be used, the amount of required model transformations as well as the difficulty of managing simulations, tools and cases.

3.3.1. Co-operative MiL simulation. In the co-simulation approach, the parts of the closed-loop system, plant model and control system model are simulated in different simulation engines. An obvious advantage of the approach is the possibility (freedom) to choose a plant modeling (simulation) language that is not possible to process with transformations. The language must be supported by a simulation engine that can be connected to another engine (simulating the control system model). With such an engine it is, however, likely that the plant simulation model can be re-used in possible later, platform specific simulations.

If a product of the MDE process is simulatable, co-simulation may also be possible to implement without additional model transformations. However, if the MDE process utilizes, for example, UML (or any non-simulatable language), simulating a modeled control solution requires transforming the solution to a simulation language. So, whether the co-simulation approach reduces the amount of required model-transformations is also dependent on the modeling languages used in the MDE process. Accordingly, the reliability of simulation results may or may not be dependent of the correctness of a simulation transformation. Support for re-using libraries requires in the approach re-usability either in the modeling tool or in the transformation that may be required for simulating design models.

Technically the co-simulation approach may be the most complex one. Values and states of connected variables and engines as well as simulation commands, e.g. running, freezing, stepping, replaying and working in slow and fast modes need to be replicated to both (or more) environments. In addition, if simulations are to be used to evaluate the behavior of the controlled system in several simulation cases, e.g. in several operation points or exceptional situations, initial values for the simulation cases must be managed for all the used simulation engines. Consequently, depending on the intended use of simulations, we argue that the co-simulation approach

may lead to difficulties in managing all the required simulation cases and information related to them.

3.3.2. Transforming plant models. Compared to the co-simulation approach, the obvious benefit of the approach of transforming plant models is that simulation cases and initial values to evaluate the control system in different situations have to be defined only for one engine. There is also no need to replicate simulation commands and states of several simulation engines.

In the approach, the plant models do not necessarily have to be simulatable but they must be possible to process with model transformations. However, a model transformation is inevitably necessary for transforming the plant model to the modeling language used in control system models. The correctness of simulation results is also dependent on the correctness of the transformation. On the other hand, either the language used in the MDE process to develop the control system/application model has to be simulatable or an additional (second) transformation is required to create a control system simulation model before transforming the plant model. In practice, this would mean two additional model transformations. Without an additional transformation, UML and related MOF-based, non-simulatable languages (e.g. SysML) would be out of the question. Support for (simulation) libraries could be in the approach implemented by supporting models of implementation blocks – or by using libraries in the second simulation transformation.

In practical implementations of the approach, see [11] and [13], the control application modeling language has been IEC 61499 which is simulatable. However, since IEC 61499 is close to PLC programming languages, the approach is also close to the approach of using SiL simulations after generating code. For example in [13] several model transformations are already used before obtaining IEC 61499 models. Lastly, a concern with the approach is that although IEC 61499 is simulatable, genuine simulation tools dedicated to simulation, instead of distributed control software development, may still be more suitable for managing and executing complex simulations than software development tools are. This concern, however, is also related to the co-simulation approach if a simulation transformation is not used.

3.3.3. Transforming control application models. Compared to the co-simulation approach, the approach of transforming control application models, again, requires a model transformation. The approach also provides the same advantages than the approach of transforming plant models. Simulation cases have to be defined only once and there is no need to replicate simulation states, values and commands.

However, compared to the approach of transforming the plant models, this approach does not suffer from the need to develop control system models with a simulatable language. Consequently, for example, UML and SysML can be used in addition to languages such as IEC 61499. Correctness of the results of simulations is dependent on the correctness of the simulation transformation, however, as we have shown the approach enables re-use of simulation blocks [1] which can be seen as a means to improve the reliability of simulation results. Lastly, the closed-loop simulation can be executed in a genuine simulation tool supporting the language used to develop the plant model, if such a modeling language has been selected for plant modeling.

Practical restrictions for selecting modeling and simulation languages apply though. In order to apply the approach, both the control system and plant modeling languages must be possible to process with model transformations. The modeling language used in the plant model must also be simulatable. However, both plant and control system models must be possible to process with model transformations also in the case of transforming plant models - in order to create and append plant models to control system models. On the other hand, if control systems are developed using MDE techniques, the control system models must be processable with model transformations in any case so that this restriction applies also to the other approaches. The restrictions of the approach of transforming control application models are thus tighter than in case of co-simulation but related to plant modeling only.

3.3.4. Summary of the comparison. A summary of the results of comparing the alternative approaches to obtain MiL simulations in MDE of industrial control applications is presented in table 1. In the table, co-sim. refers to co-simulation, TCSM to the approach of Transforming Control System Models and TPM to Transforming Plant Models. CSML and PML refer to Control System and Plant Modeling Languages.

Co-simulation places least restrictions on possible modeling and simulation languages because plant models do not necessarily have to be accessible with model transformations. However, simulation technically the approach is the most complex one, it may lead to difficulties in managing simulation cases and it either requires the control system modeling language (CSML) to be simulatable or an additional model transformation for creating one. If a transformation is not developed, it may require the control system to be simulated in a software development tool, instead of a simulation tool. However, in this case the results of simulations are not dependent of correctness of simulation transformations, as is the case in the transformation-assisted approaches.

The two transformation-assisted approaches do not require coupling simulation engines, states or commands. However, the approach of transforming plant models is more restricting than the approach to transform control system models. Either the control system model must be simulatable or the control system model must be first transformed to a simulatable form, with an additional (second) model transformation. In addition, if a transformation is not developed, the approach may

require the control system to be simulated in a software development tool, instead of a simulation tool. All the approaches enable re-use of simulation blocks, either at modeling level or with use of simulation block libraries. Consequently, this feature is not included in the table. It is also natural that if a simulation approach requires a transformation, the correctness of the results is dependent on correctness of the transformation. Thus, this feature has not been repeated in the table, in addition to listing the numbers of additional transformations required in the approaches.

Table 1. Results of comparing alternative approaches to MiL simulations in MDE in the automation domain

	co-sim.	TPM	TCSM
Enables use of genuine simulation tools	?/X	?	X
Simulation tools must be connectable.	X	-	-
Number of additional transformations	0/1	1/2	1
CSML must be simulatable	X/-	X/-	-
PML must be simulatable	X	-	X
Additional work with simulation cases	X	-	-
Additional simulation management work	X	-	-
PML must be transformable	-	X	X
CSML must be transformable	X	X	X

4. Observations from applying the approach

In our work, the objectives of developing the simulation integration have been early, development-time simulation of applications and the resulting capabilities to test, compare and prototype control applications and functions. Design flaws should be also corrected as early as possible so that they would not affect adversely on subsequent design phases. Naturally, this requires prompt feedback about the designs that can be in the automation domain acquired with simulations.

The approach to create simulation models from UML AP models is transformation-assisted and has been developed and published gradually. We started from the core concept and support for librarized simulation blocks [18], developed support for using logic diagrams to model and simulate application specific blocks, e.g. interlockings [1], and lastly focused on automation sequences [3] that are necessary in shifting between operating points and for start-up and shut-down analyses. In detail, the approach has thus been presented in [18], [1] and [3]. With respect to the possible MiL simulation approaches in MDE that were discussed in section 3, the approach falls to the *MiL* category using *single simulation engine* and transforming control system models to plant models. The approach also enables the use of *libraries of existing simulation blocks*.

4.1. Simulation experiments

So far, the simulation approach has been used in 3 published or to-be-published simulation experiments that have covered both process industry as well as machinery processes. On the other hand, the experiments have covered different types of control functions that are needed in the domain. Basic feedback control, firstly, has been utilized in all the simulation experiments. Interlockings, secondly, have been used to enable and disable devices and to restrict set-points of controllers in both machinery- and process-industry-related simulations. Sequences and binary controls, lastly, have been used in the latest simulation experiment related to batch production of pulp.

In the first simulation-related publication [18], we modeled and simulated a control application for a heated water supply system. The control application consisted of three control loops, each containing an interlocking and controlling one of the controlled variables of the process that is presented in figure 2. The simulation counterpart of the control system was in this experiment composed solely of librarized simulation blocks that were coupled together and parameterized by the transformation according to the control system model.

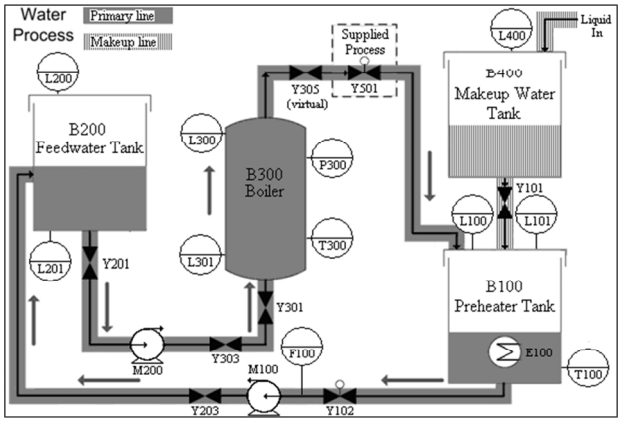

Figure 2. The heated water supply system used as an example process in [18]

In [1] the process to be controlled was a mechanical cart system that is presented in figure 3. The cart system was moved with an electrical motor, stopped with brakes and must be kept outside forbidden areas regardless of the set-point from a higher level control system. The control system model contained both librarized blocks and application specific interlocking blocks. Alternative solutions to keep the cart in its allowed area regardless of the value of the non-restricted set-point were defined with a logic diagram implementation of our tool and evaluated by using the transformation to simulation models and simulating the designs.

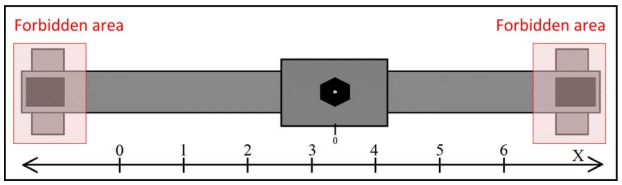

Figure 3. The mechanical cart system used as an example process in [1]

Lastly, in [3] the example process was a batch process system, the piping and instrumentation diagram of which is presented in figure 1 in section 1. In this experiment, in addition to feedback control, we modeled also a control sequence that was used to control the execution of the batch and to control individual devices. Compared to the other simulation experiments, the total process model was also the largest that we have used, so far. Including both the equations caused by the control system and those of the plant, the total amount of equations in the model was over 1400. (Modelica is an equation based language.) Of these equations, a large proportional were trivial; however, the plant model alone still contained approximately 50 differential equations and several hundred complex algebraic equations.

4.2. Benefits of early MiL simulations

In the simulation experiments we have covered all the common aspects of basic control systems: binary control, feedback control, sequential control as well as interlockings. Based on the experiments, they can be also used concurrently. For example, set-points of feedback controllers as well as binary control commands can be given in sequences concurrently to interlockings that enable and disable devices and controllers.

On the other hand, in addition to processes of different industries, the modeling and simulation approach has scaled well to processes of industrial size and complexity. In the latest modeling and simulation experiment, the total amount of equations related to the closed-loop system was over 1400. Although such an amount of equations would not necessarily enable modeling of large processes such as paper machines, it would enable quite detailed modeling of partial process industry processes or mechanics of machinery systems.

With respect to benefits of applying simulations, in [18] and [1] we prototyped several different interlocking and control solutions. Related to interlockings, the latter publication contains a comparison of two solutions and results of their simulations. Tunings of e.g. PID controllers to obtain acceptable dynamic behavior have also been searched for all the published experiments. As such, the approach has shown to enable prototyping and comparing alternative approaches and tunings to implement control and interlocking functions.

Related to revealing inconsistencies and missing requirements, we have been able to notice shortcomings in requirements and implementations. Especially, related to the batch process experiment [3] we developed the control solution and sequence step by step and simulated the phase products. For the incomplete phase products, the simulation results clearly indicated missing parts of the sequence as well as controllers which led to stopping of the execution of the batch sequence or temperatures and pressures to rise above their desired values.

Lastly, according to our results, MiL simulations can be useful also in testing exceptional situations. Activations of interlockings due to hazardous set-points have been successfully tested in several experiments including the ones published in [18] and [1]. Safety studies related to changes in the physical processes could be implemented easy as well. For example jams of valves or motors could be tested by presenting the timed changes in dynamics in the process models while using the simulation approach in a normal way since the control application parts are only added to the process model. As such, we regard the MiL approach suitable also for safety studies which was one of the general benefits of simulations mentioned in [16] and [17].

To draw conclusions on the experiments, we have demonstrated the suitability of the approach to processes of different industries and of industrial size. In our experiments, the simulation approach has been found useful in comparing and prototyping alternative control and interlocking approaches, testing sequences as well as finding acceptable controller tunings. Simulations have helped finding missing implementations and requirements as well as testing exceptions that could be dangerous to test with actual physical processes.

5. Conclusions

In this paper, we have discussed and compared approaches to integrate simulations to model-driven engineering of industrial automation and control applications. If simulations are used in an MDE process, they should follow the Model-in-the-Loop approach (MiL). Other types of simulation are supported by PLC and DCS platform vendors and are applicable after generating code. Because applications are already often composed of re-usable implementation blocks, re-use should be also enabled in simulations in order to improve the reliability of the simulations.

The conclusions of comparing MiL simulations to using target platform specific simulation tools were that most of the general benefits of use of simulations can be also obtained by integrating early MiL simulations to an MDE process. The restrictions of early MiL simulations are related to missing hardware, which may complicate, for example, operator training. However, without hardware, MiL simulations can be performed prior to choosing target platform and performing hardware design. MiL simulations could also be used in companies performing out-sourced development tasks and it may be possible to re-use plant models in later simulations, too.

We also compared the conceptual approaches to implement MiL simulations with the conclusion that all the approaches have both benefits and disadvantages over each other. Co-simulation, firstly, does not require model-transformations provided that the control system modeling language is simulatable. However, it requires more work in configuring simulation cases and engines. Of the transformation assisted approaches, transforming plant models may be less restrictive related to modeling languages. However, it requires an additional model transformation if the control system modeling language used in the MDE approach is not simulatable and may lead to using software development tools to system simulation, instead of genuine simulation tools.

The simulation approach of the authors falls to the MiL category and uses model transformations to create and append control application specific parts to the plant models. According to results obtained with three published simulation experiments, the approach suits for both machinery and process industry applications. It has also been used beneficially in simulation of a large process industry process. The general benefits of simulations obtained so far are similar to those anticipated by other researchers. Our approach has additionally enabled prototyping, experimenting and comparing control and interlocking solutions, searching controller tunings and detecting inconsistencies in requirements and design. Related to general benefits of simulations, we have also used MiL simulations to evaluating control and interlocking solutions during exceptions. The simulations have been possible to implement early and without either the physical processes or the expensive control system hardware.

References

[1] T. Vepsäläinen and S. Kuikka, ”Simulation-Based Development of Safety Related Interlocks”, *Simulation and Modeling Methodologies, Technologies and Applications*, Springer Berlin Heidelberg, 165-182, 2013.

[2] D. Hästbacka, T. Vepsäläinen and S. Kuikka, ”Model-Driven Development of Industrial Process Control Applications”. *Journal of Systems and Software*, Vol. 84, No. 7, 1100 – 1113, 2011.

[3] T. Vepsäläinen and S. Kuikka, “Simulation Assisted, Model-Driven Development of Automation and Control Applications - Modelling and Simulation of Control Sequences”, *Control Engineering Practice*, 2013. (SUBMITTED)

[4] H. Shokry, and M. Hinchey, “Model-Based Verification of Embedded Software”. *Computer*, 53-59. 2009.

[5] J. Carrasco and S. Dormido, “Analysis of the Use of Industrial Control Systems in Simulators: State of the Art and Basic Guidelines”. *ISA Transactions*, Vol. 45, Issue 2, 295-312, 2006.

[6] A. Plummer, “Model-in-the-Loop Testing”, *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, Vol. 220, No. 3, 183–199, 2006.

[7] H. Chae, X. Jin, S. Lee and J. Cho, “Test: Testing Environment for Embedded Systems Based on TTCN-3 in SILs”. *Advances in Software Engineering*, Springer Berlin Heidelberg, 204–212, 2009.

[8] M. Short and M. J. Pont, “Assessment of High-Integrity Embedded Automotive Control Systems Using Hardware in the Loop Simulation”, *Journal of Systems and Software* Vol. 81, Issue 7, 1163 – 1183, 2008.

[9] M. Schlager, R. Obermaisser and W. Elmenreich, “A Framework for Hardware-in-the-Loop Testing of an Integrated Architecture”. *Software Technologies for Embedded and Ubiquitous Systems*, Springer Berlin Heidelberg, 159–170, 2007.

[10] G. Stoeppler, T. Menzel and S. Douglas, “Hardware-in-the-Loop Simulation of Machine Tools and Manufacturing Systems”. *Computing & Control Engineering Journal, Vol.* 16, Issue 1, pp. 10–15, 2005.

[11] C. Yang and V. Vyatkin, “Transformation of Simulink Models to IEC 61499 Function Blocks for Verification of Distributed Control Systems”. *Control Engineering Practice*, Vol. 20 No. 12, 1259–1269, 2012.

[12] V. Vyatkin, H.-M. Hanisch, C. Pang and C.-H. Yang, “Closed-Loop modeling in Future Automation System Engineering and Validation*”, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, Vol.* 39 No. 1, 17 –28, 2009.

[13] I. Hegny, M. Wenger and A. Zoitl, ”IEC 61499 Based Simulation Framework for Model-Driven Production Systems Development“, *Emerging Technologies and Factory Automation, IEEE Conference on*, 1–8, 2010.

[14] L. Ferrarini and A. Dede, “A Model-Based Approach for Mixed Hardware in the Loop Simulation of Manufacturing Systems”, *10th IFAC Workshop on Intelligent Manufacturing Systems*, 41–46, 2010.

[15] M. Karaila and T. Systä, ”On the Role of Metadata in Visual Language Reuse and Reverse Engineering – An Industrial Case”, *Electronic Notes in Theoretical Computer Science*, Vol. 137, Issue 3, 29-41, 2005.

[16] J. Dougall, “Applications and Benefits of Real-Time I/0 Simulation for PLC and PC Control Systems”, *ISA Transactions*, Vol. 36. No. 4, 305-311, 1998.

[17] T. Karhela, ”A Software Architecture for Configuration and Usage of Process Simulation Models: Software Component Technology and XML-Based Approach”, *PhD Thesis,* VTT Technical Research Centre, 2002.

[18] T. Vepsäläinen, D. Hästbacka and S. Kuikka, “Simulation Assisted Model-Based Control Development - Unifying UML AP and Modelica ML”, *11th International Middle Eastern Simulation Multi-conference*, 43-50, 2010.

Publication 6

Vepsäläinen, T., Kuikka, S. (2011) Towards Model-Based Development of Safety-Related Control Applications. Proceeding of the 16th IEEE International Conference on Emerging Technologies and Factory Automation. Toulouse, France, September 5-9, 2011, pp. 1-9.

DOI: 10.1109/ETFA.2011.6058979

Towards model-based development of safety-related control applications

Timo Vepsäläinen
Tampere University of Technology,
Department of Automation Science and Engineering
P.O. Box 692, FIN-33101 Tampere, Finland
timo.vepsalainen@tut.fi

Seppo Kuikka
Tampere University of Technology,
Department of Automation Science and Engineering
P.O. Box 692, FIN-33101 Tampere, Finland
seppo.kuikka@tut.fi

Abstract

Model-based techniques have been recently the topic of numerous publications in different domains. In addition to producing revised models and executable applications, model-based techniques could also aid the understandability of design and consistency between design artefacts. These properties are also focal to development of safety-related applications, in addition to the ability to produce documentation about the systems. In this paper, we strive to create a new model-based approach for development of safety-related applications by integrating risk analysis techniques and modeling notations from several related languages and standards. The notations and their characteristics are compared to the requirements of the vital functional safety standard, IEC 61508, and illustrated with an exemplary modeling case.

1. Introduction

The idea of focusing to models in development of systems and software applications has recently been the topic of numerous publications in several domains, including industrial control engineering. Due to the interests and publications, there are also several acronyms related to the concept, some of which are already registered trademarks of organizations that have pioneered in utilizing and standardizing the approaches. For example, Model-Driven Architecture (MDA) is a trademark of OMG that also maintains the specifications of modelling (UML, SysML), metamodeling (MOF) and transformation (QVT) languages that can be used in conjunction to MDA.

The idea of model-based development and related approaches, e.g. MDA and model-driven engineering (MDE), is to use models as primary engineering artefacts during the development of applications. In the domain of systems engineering, model-based systems engineering (MBSE) refers to applying models as part of the systems engineering process in order to support analysis, specification, design and verification of the system being developed [1]. However, in systems engineering, the main focus of model-based methods may not always be in producing more accurate models or executable applications based on models but also to aid the analysis, understanding and documentation of the systems. In development of software applications, it has been more natural to target to the ability to automatically utilize models and specifications in production of revised models and executables.

The authors of this paper have taken part in development of an approach, and tools supporting the approach, to automatically utilize models in development of industrial control applications. The approach developed during the AUKOTON project has been presented in detail in [2] and consists of three modelling phases during which the requirements, functionality and platform specific details are specified. The modelling concepts used during the development are based on UML automation profile (UML AP) [4] and enhancements to the profile developed during the AUKOTON project. The assessment of industrial applicability of the AUKOTON approach has been presented in [3].

However, more attention could be paid on analysis of models and producing documentation (potentially for an in-house design knowledge repository) about the applications. In some model-based development approaches, documentation may not have been a focal asset also because of some common trends in software engineering. For example, agile development processes are focusing on the ability to react to changes and value working software above extensive documentation. These purposes are probably beneficial in development work that is based on changing user requirements.

In development of mission-critical and safety-critical systems and applications, requirements are based on hazard and risk analysis and are not likely to change that often. In addition, documentation is still of importance because of the need to be able to prove characteristics of the systems. Development of safety-critical applications (conforming to safety standards) is required to produce a vast amount of documentation about the systems and the development activities used. It is thus an interesting question, whether or not also development of safety-

critical applications, requiring thorough analysis and documentation, could benefit from application of model-based techniques.

Whereas safety-critical applications are often quite small compared to, for example, ordinary industrial control applications, in the development of them there might not be need to automate processing of bulk information. Instead, the development of safety-critical applications could benefit more from aiding understandability of the system being built, the hazards related to the system, traceability between design artefacts and the ability to run analysis model-checks and generate documentation about the system. In this paper, we discuss the possibilities to support development of safety-related software applications with model-based development (MBD) techniques by extending our AUKOTON approach. Specifically, we focus on modelling of information that supports the understanding of hazards, safety requirements specification, and detailed design.

The development process and essential *requirements* of the vital functional safety standard, IEC 61508 [5], are discussed in section 2. Section 3 focuses in modelling needs and possible notations to be used. In section 4, we discuss the concept of risk and hazard modelling. Section 5 focuses on requirements modelling. In section 6, before presenting related work and concluding the paper, we develop an exemplary modelling project utilizing the developed modelling concepts for the physical process presented in figure 1.

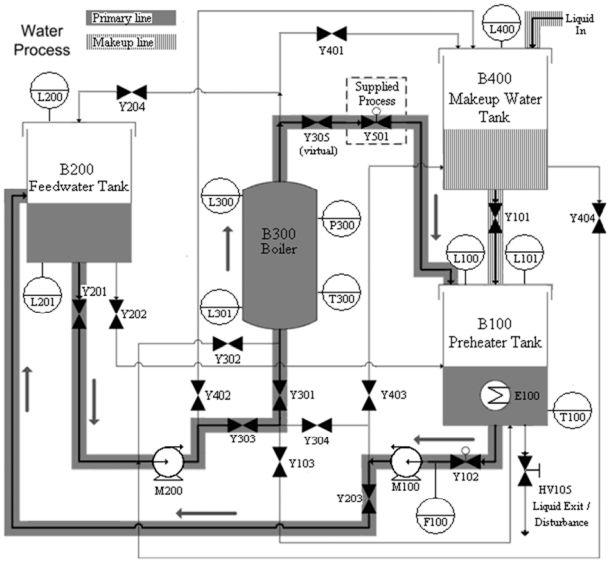

Figure 1. An exemplary physical process for the modeling example in section 6.

2. Development of safety critical systems

IEC 61508 is an international standard of functional safety of electrical/electronic/programmable electronic safety-related systems. The standard consists of seven parts that focus on different aspects of development of safety-related systems including, for example, general requirements and software requirements. The current version of the standard has been published in 2010. [5]

IEC 61508 is of special importance as a functional safety standard because of several reasons. Firstly, the standard has been renewed a short while ago. Consequently, for example, the list of recommended actions and techniques should be as modern as applicable. Secondly, one of the purposes of IEC 61508 is to facilitate development of industry specific standards, which increases its importance. Such industry specific standards include, for example, IEC 62061:2005 in machinery. Finally, according to our interviews during the Ohjelmaturva project with several Finnish and international companies in mobile working machines domain, IEC 61508 may be the most difficult safety standard to be used in the development. Partly this is because of the vast amount of requirements and techniques, partly because of the difficulty of applying some of the required techniques.

According to the standard, the lifecycle of safety-related system starts from concept definition. Concept definition is followed by overall scope definition, hazard and risk analyses, overall safety requirements specification, safety requirements allocation, planning phases, realization phases of safety related systems of various implementation techniques, and so on. The development of software parts of systems is covered mainly in the part 3 of the standard. It includes software safety requirements specification, validation planning, design and development, integration of software and hardware, operation and maintenance procedures specification, and validation. [5] The standard is built so that the most natural way to fulfill the requirements would be to utilize the traditional V-model development process. However, the standard allows any development process to be used, provided that the requirements of the standard are fulfilled. Thus, also agile processes could be used, but, according to [6], problems may arise because of lack of ability to provide necessary documentation.

IEC 61508 is a risk-driven standard. After scope definition, the actual development process starts from identification of the hazards and estimations of risks, followed by specification and allocation of requirements and then proceeds towards implementation. The latter phases of the process are built on information produced by the former phases. To the different development phases, the standard suggests means to make sure that the actual risks are taken care of and that the system will function in a proper way. Traceability requirements ascertain that the risks and hazards are the basis for development of the safety-related functions - the process is thus risk-driven. Recommended techniques, on the other hand, ascertain that the functionality is specified and implemented correctly.

2.1. Traceability

In IEC 61508, a repeating requirement for phase products of safety-related system development is the consistency and traceability between them. For example, system level requirements must be both backward traceable to the perceived safety needs and forward traceable to software requirements. Architecture design must be traceable to safety requirements, software design must be traceable to safety requirements and design specifications must be traceable to test specifications. [5]

In traditional, document-based development of systems and applications, the traceability between the development artifacts may be difficult to fulfill. For example, traceability between identified hazards and software safety requirements could be supported by specifying (explicitly) the unique identifiers of the hazards that the requirements have been specified for. Similarly, requirements could be linked to software safety functions and architectural decisions by specifying the IDs of them. In addition, hyperlinks between the specifications in same or different documents could be defined to aid the discovery of the artefacts related to each other. However, documents are still difficult to keep up-to-date when something is changed and they don't support impact analysis. In addition, generation of any kind of summary about the traceability, such as a *traceability matrix*, or searching for hazards that have not been addressed by any requirement, would be very difficult.

2.2. Correctness, completeness and unambiguousness

Some other repeating requirements for different phase products of safety-related software development are, according to IEC 61508, correctness, completeness and unambiguousness. To achieve the correctness and freedom from faults, the standard recommends, for example, application of formal or semi-formal techniques during, for example, requirements specification and detailed design phases. However, even the standard admits that application of formal methods may complicate the achievement of the understandability requirement. [5]

Indeed, formal methods are, according to the knowledge of the authors, often not familiar to developers of software applications in industrial control and machinery. However, in order to develop safe systems, the requirements need to be specified strictly and in an unambiguous manner. Instead of formal methods, the design specifications could be also based on semi-formal, domain specific concepts, that would fulfill the requirements of the standard. In the domain of industrial control, logic diagrams have been traditionally used for specification of control approaches and safety-function-like interlockings. Such specifications present requirements but also, in some cases, detailed design and are both familiar to developers and, at least, a semi-formal approach.

Specifying requirements has been found difficult in software engineering in general. Author of [7] has analyzed the quality of produced software in about 12500 projects from year 1984 to 2008 and the defects delivered (and removed) during the projects. According to the survey, in best-in-class-quality, a main portion of defects delivered were related to defects in requirements specification, partly because defects in requirements are difficult to discover.

One effective means to aid and improve requirement specifications could be the use of inspections. In model-based development, inspection-like activities could be supported by presenting both the hazards and requirements in same models than design and by linking them together. By doing so, the developers of the system could always follow the requirements to the hazard model when in doubt. In addition, for example, definers and implementers of safety functions could be instructed to always check the consistency between the artefacts before writing a single line of specification or code. By doing so, the consistency would be checked several times during the development by different people, including developers of safety requirements, implementers and testers, just to name a few. Visibility could thus aid both correctness and completeness.

3. Model-based approach to safety

In MBD, models are used as primary development artefacts instead of, for example, documents. In MBD of safety-related applications, the applications should, accordingly, be developed by utilizing models but also considering the requirements of safety standards. Thus, when applying MBD techniques to development of safety-related systems, attention should be paid on the documentation needs. In our opinion, the special needs are the properties discussed in 2.1 and 2.2: traceability, correctness, completeness and unambiguousness.

Traceability, in this case, refers to traceability between all the phase products such as hazards, requirements, and detailed design artefacts. In software engineering, the most used modelling language is UML. However, UML does not address the traceability explicitly with any modelling concept. In contrast, a UML profile SysML defines concepts for defining relations (traces) between requirements and design artefacts and between requirements and test cases. These concepts do not cover all the traceability requirements of IEC 61508 but form a foundation that could be further extended. However, supporting *traceability* would also require including more concepts, such as hazards in the scope of modeling.

In addition to traceability, there are several reasons to add hazards to the scope of models. As discussed in 2.2, the hazard information could be made visible and available for the developers in order to aid the

understandability of the requirements and design. Consistency between design artefacts could be inspected by different developers to aid the *correctness* and *completeness* of requirements. Similarly to traceability, hazards are not covered by UML. There are, however, reported approaches to cover structured presentations of hazards, such as, the safety analysis profile [8] and the approach of the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoSFT) [9].

Requirements specification may be the most critical part of development of complex systems. Requirement specifications should, on one hand, be formal enough to fulfil the requirements of standards but also be based on concepts familiar to developers. However, for example UML defines only use case concept for stating (functional) requirements. In addition, SysML defines a set of textual requirement specification concepts but they can hardly be characterized as formal. In the domain of industrial control, there are also standards related to functional requirements, including IEC 62424 [14] and IEC 61804 [12]. These standards are more familiar to the developers in the domain but also enable structured presentation of required functionality and coupling to the instrumentation of the system.

In our approach, we pursue to integrate modeling notations from several languages in order to facilitate the development of safety-related software applications with model-based techniques. In more detail, we aim to enhance traceability, correctness, completeness and unambiguousness of design with models. The modeling languages and notations of interest include: UML, SysML [11], safety analysis profile [8], QoSFT profile [9], IEC 62424 [14] and IEC 61804 [12]. These languages and notations will be next discussed from the point of view of modeling hazards, requirements and detailed design with the aim of collecting practices to be used in modeling in conjunction to UML AP.

4. Risk and hazard modeling

Currently, modeling of hazards and risks is not covered by many modeling languages or profiles in software engineering. In the approach of [10], the focus is in incorporating safety requirements in software architecture and evaluations of the architectures based on safety analysis methods. The developed metamodels include FMECA (Failure mode, effects and criticality analysis) and FTA (fault-tree analysis) metamodels.

In QoSFT profile [9], the main objective of the modeling may not be in detailed specification of how the hazards may occur. Instead, it is focused on factors determining the magnitudes of risks (likelihood, consequences), compromised assets, stakeholders, and the treatment of risks. Treatment approaches include: avoiding risk, reducing its likelihood or consequences, and retaining and transferring the risks. Tracing of risks to requirements is not covered by the profile.

In safety analysis profile [8], both the occurrences of hazards and tracing hazards to requirements are covered. To the definition of occurrences of hazards, the profile suggests the use of FTA that can be also used in a quantitative way. Another benefit of FTA is that it can aid the design of safety functions. Safety functions can be designed to disjoint the fault or event sequences leading to the hazards so that for the hazards to occur, also the safety functions would need to fail.

To support the documentation of hazards, we suggest a combination of the modeling notations. FTA, suggested by both [8] and [10], is a very analytic technique and also enables quantitative analysis of hazards. To support traceability, it should be possible to trace hazards to requirements. However, in addition to tracing the hazards to requirements, it would be beneficial to document the approach to handle the risks, similarly to the approach of QoSFT profile.

The following (partial) metamodel, presented in figure 2, has been defined to fulfill the above mentioned needs. However, properties supporting quantitative analysis, such as, mean time between failures (MTBF) and for example logic operators supporting FTA, such as, OR and NOT, have been left out of the figure. The hazards and event sequences leading to hazards can be presented with FTA elements: Hazard, Required and Resulting Conditions, Fault and logical operations (not shown in the figure). Risk treatments specify the approaches to treat the risks and link them to requirements. Hazards can be related to each other with different kinds of relations and Faults can be linked to modeled hardware (SysML) or software (UML) elements causing the fault.

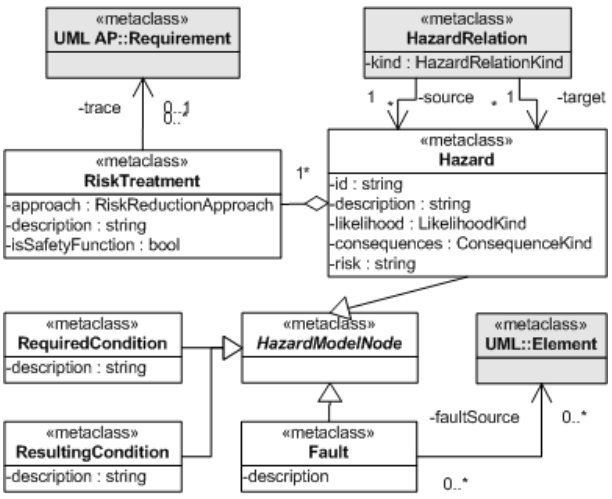

Figure 2. Part of the developed hazards metamodel.

5. Requirements and design

As presented earlier UML does not contain concepts for stating explicit requirements. Use cases can be used for presenting interactions between users and systems but they usually require additional textual descriptions to enable specification of what exactly happens. In addition, other modeling concepts, such as classes and state machines, can be used in analysis phases but the diagram point of view is often in solutions instead of requirements. By use of SysML [11], requirements can be specified with requirement concepts that include text and id attributes. Traceability of requirements is in SysML supported by traces that can be used for tracing requirements to implementing elements and test cases. Traces can also be searched from models, for example with model checkers, in order to generate traceability documentation or to search for overlooked requirements. However, SysML requirements can hardly be characterized as formal.

In the industrial control domain, there are at least two standards that address the functional requirement specification issue: IEC 62424 [14] and IEC 61804 [12]. IEC 62424 defines a specification for presentation of requirement-like process control engineering (PCE) requests in piping and instrumentation diagrams (P&ID) and enables data exchange between P&ID tools and control engineering tools in order to optimize the engineering process. The standard also allows identification of requests (requirements) that are related to safety and definition of the corresponding SIL or PL levels. Another advantage is the direct linking to devices and instruments of the process. However, IEC 62424 does not allow definition of safety function logic that would be advisable from the point of view of IEC 61508.

IEC 62424 is also part of AutomationML which is focused on data exchange and the integration of engineering disciplines in development of manufacturing systems. The information addressed by the language includes manufacturing system topology, geometry, kinematics, and control behavior. [15]

IEC 61804 originates from power plant industrial sector and aims to utilize IEC 61499 function blocks (FB) for specification of functional requirements. Process flow diagrams are first used to identify the process elementary operations. The required (control) functions are then identified and marked in P&I diagrams, structured to sets and presented in control hierarchy diagrams. Finally, the details of the required functions are specified by using a vendor neutral (but unambiguous) FB language. [12] Similarities between 61804 and 62424 include, at least, tight integration to instrumentation, which could aid the understandability of requirements. Neither standard, however, addresses the traceability to implementation although shifting to implementing function blocks may be straightforward.

In UML AP, requirements are structured concepts with attributes for id, description, priority, rationale and source. Concepts are divided to a hierarchy based on the basic viewpoint, such as, to interface with sensors or to interlock devices. The required information interchange between required functions can be modeled with port-like requirement interfaces. By connecting safety and control function requirements to requirements presenting needs to interface with the instrumentation, the interfaces also enable modeling of the integration to instrumentation. Requirements can be traced to implementing model elements with requirement traces (similar to the traces of SysML). To enable definition of safety requirements, it was seen that support for modeling of unambiguous interlocking logic (like in IEC 61804), required SIL (or PL) levels and allocation of requirements to different implementation techniques would be needed.

Figure 3 presents part of the additions to requirement metamodel of UML AP, detailed diagrams of which are presented in [2]. In UML AP, requirement refinements are used to define additional information to requirements. In case of safety requirements, such refinements could be related to required SIL (or PL) level, or in early stages of requirements specification, the (allocated) implementation technology for the requirements. Logic operation library was also defined to enable the definition of exact logic. The logic concepts are currently applicable in *internal block diagrams* that can be used to depict internals of requirements in our tool environment.

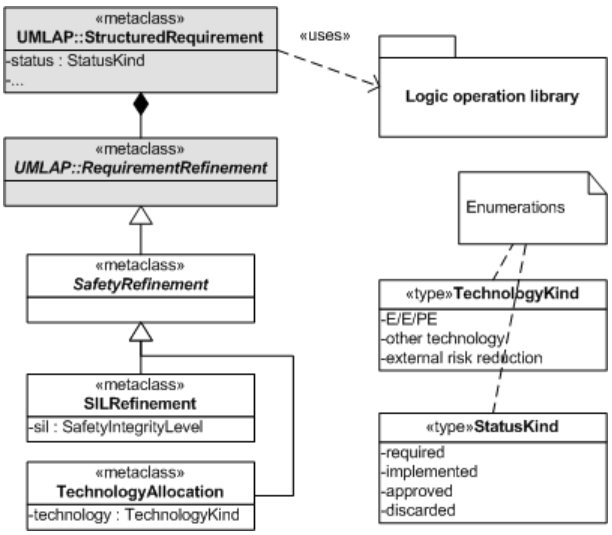

Figure 3. Part of the additions to the UML AP requirements sub-profile.

In this paper, our main focus has been in modeling hazards and requirement information. The reason is that, in our opinion, those aspects are the ones that are difficult to cover in modeling with current languages. However, commonly used modeling languages, such as,

UML could be used in detailed design at least if the system is to be implemented with general purpose programming languages. UML can be seen as a semi-formal method so it also fulfills the requirements of IEC 61508 (table B.7 of part 3). In addition to detailed design, UML and SysML are suitable for depicting software and system architectures with, for example, component, block and deployment diagrams.

The functional modeling concepts of UML AP, Automation Functions, are related to function-block-based development. FB languages (with defined subsets) are also highly recommended to all SIL levels by IEC 61508. The current version of the profile also enables detailed definition of interlocking and safety function logic, which was seen to be missing in the assessment of the previous version [3]. To achieve this, we have enabled the use of logic library that was discussed earlier also in specification of inner logic of Automation functions. The justification is that logic diagrams are closely related to (highly) recommended programming languages, fulfill the required formality, are familiar to developers and thus aid unambiguity.

6. A modeling example

The purpose of this section is to provide an exemplary utilization of the developed modeling concepts. In the following subsections, we present a model describing hazards related to a simple process and how the modeling concepts can be used in specification of requirements and traceability between the hazards and requirements.

6.1. Hazard model

The system of interest is visualized in figure 1. The system constitutes a simple, closed system providing hot pressurized water for the supplied process, and consists of two storage tanks for hot liquid, a boiler, two pumps for pumping the water between the tanks, and a few valves and sensors. The system is capable of causing harm to the environment, at least, by overflowing from the tanks B100 and B200 and by leaking from the pumps, which may be caused by damaging the pumps by running them dry.

The occurrences of hazards of overflowing of tank B200 and running pump M100 dry are presented in figure 4 (drawn with the UML AP Tool). In this case, the occurrences of the hazards do not require faults in the system. Instead, overflowing of B200 requires that water is pumped to the tank with the tank being already full. Pump M100 running dry requires that the pump is used while B100 is empty. Both hazards are traced to interlocking requirement I-M100 (Interlock for M100), as indicated with the risk treatments, with the goal to *avoid* the risk. Details of the hazards: likelihoods, consequences and resulting risk values are not shown in the simplified figure below.

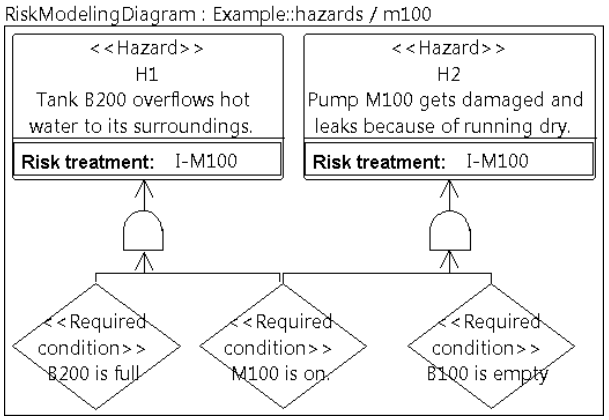

Figure 4. Hazards related to tank B200 and pump M100.

In a similar way, overflowing of tank B100 can be caused by pumping water to it (through boiler and supplied process) while the tank is already full and the pump M200 running dry by using it while tank B200 is empty. However, another way in which tank B100 can overflow is to drain additional water to it with use of valve Y101 when the tank is full. The occurrences of these hazards that can be treated by interlocking pump M200 and valve Y101 are shown in figure 5.

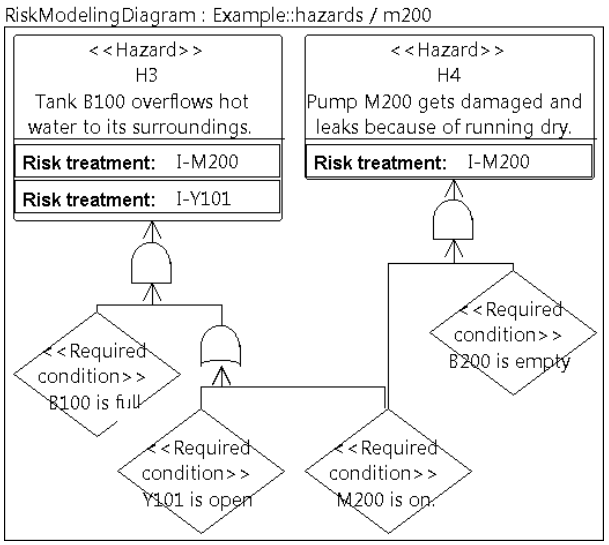

Figure 5. Hazards related to tank B100 and pump M200.

6.2. Requirements

The interlocking requirement related to pump M100 is presented in its context and with details in figures 6 and 7, respectively. As seen in figure 6, the interlocking requires information about the levels of water in tanks B100 and B200, so the requirement is related to the

corresponding measurement requirements and also to the requirement of controlling pump M100 (not shown in the figure). Because of relatively small amount of energy handled in the process, the SIL levels for the required functions were set to 0 which means that they are non-safety-critical interlocking requirements. Actual, required interlocking logic for pump M100 is presented in figure 7: the pump is locked (disabled) if the water level in B100 is below 0.1 or the level of water in B200 is over 0.9, so that the condition sequences in figure 4 are disjoint. The notation in figure 6 is discussed in more detail in [2].

Similarly, interlocking for the pump M200 can be required to stop the pump if the water level in B200 is below 0.1 or the level of water in B100 is over 0.9. The latter condition can be used also in the interlocking for Y101 (I-Y101) to close the valve. These interlocks are not shown in the figures.

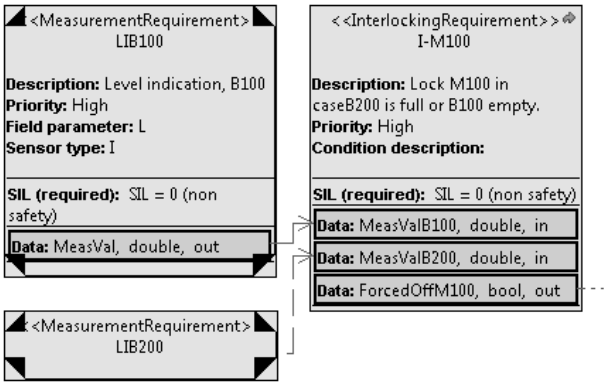

Figure 6. Interlocking requirement for M100.

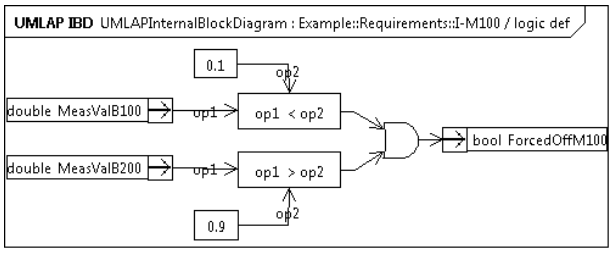

Figure 7. Detailed logic of Interlocking requirement M100.

6.3. Traceability

The example process covered in this paper is relatively small but illustrates the use of FTA and requirement concepts discussed earlier, in conjunction to UML AP. In the model, identified hazards are linked to functional requirements with risk treatments that also specify whether the requirement represents a required safety function and with the risk reduction approach (in the example: avoid). With the requirement trace concepts of UML AP [2], the requirements could be further linked to implementing (modeled) elements that could be related to both architecture and detailed design. In addition, SysML defines (verify) relations to be used between requirements and test cases. As a whole, the approach thus supports traceability between hazards, requirements, architecture, detailed design and test cases.

As the traceability is defined with explicit traces, it can also be analyzed automatically. Currently, our tool environment supports exporting *traceability matrices* (Microsoft Excel sheets) presenting the traceability between identified hazards and requirements and also between requirements and (detailed or architectural) design elements. When creating a hazard trace table, the hazards are first collected to a list. The corresponding requirements are identified and listed based on the risk treatments and finally the information content is exported to Excel sheets by adding a row for each hazard and a column for each requirement a hazard is traced to. Finally, the requirements that a hazard is traced to can be indicated by marking the columns of the requirements. In the process, hazards that are not traced to any requirements are highlighted with red color in order to warn the user about possibly overlooked hazards. An example of an automatically generated traceability matrix related to the hazards and requirements discussed earlier is shown in figure 8.

	A	B	C	D	E
1	**Hazard trace table**	InterlockingRequirement: I-M100	InterlockingRequirement: I-M200	InterlockingRequirement: I-Y101	
2	Hazard: H1	X			
3	Hazard: H2	X			
4	Hazard: H4		X		
5	Hazard: H3		X	X	
6					

Figure 8. An automatically generated traceability matrix from hazards to requirements.

Similarly to the hazard traceability matrix, requirement traceability matrices can be automatically constructed between requirements and other model elements. In the implementation, the main difference is that instead of risk treatment elements, the table is constructed based on trace requirements elements of UML AP. In addition, because the trace relations of SysML define explicitly the client and supplier elements, they could also be used in creation of similar tables. However, the use of SysML traces is not yet automated in our modeling tool.

Model checks could also be easily added to the tool environment to perform various checks. For example, requirements that are coupled together must have matching SIL refinements, all requirements must be traced to test cases modeled with SysML, and risk treatments must document the risk reduction approaches, just to name a few. This kind of checks could ease the developers work by identifying possible flaws in the models. Consequently, their main purpose could be to support the developers during the actual development work whereas the traceability information could be useful both during the development and after it. In more detail, during the development the traceability matrices could aid the understandability and inspections of design and after it by storing the valuable traceability information.

7. Related work

In addition to the approach outlined in this paper, supporting development of safety-related software with model-based techniques has been studied also by other researchers. In [16], Guillerm et al. discuss the use of SysML to address requirements definition, traceability as well as verification and validation in system engineering process. In the paper, they propose the use of UML and SysML in requirements definition and extend the languages with stereotypes related to documenting risks. However, the information model does not address the modeling of how the hazardous situations occur which would be required to understand how the required safety functions are to treat the risks, for example, by interrupting fault sequences leading to hazardous situations.

In the approach presented in [17], Biehl et al. attempt to integrate safety analysis to model-based development in automotive industry. They automate translation from EAST-ADL2 to HiP-HOPS with 2-phased transformation in order to automate performing of safety-analysis on refined models with minimal effort. However, in the approach the focus is in automating the safety analysis, not in using and understanding the information produced by safety and hazard analysis in a constructive way, as is in our approach.

The UML Profile for developing Airworthiness-Compliant SafetyCritical Software [18] intends to extract the key safety-related concepts from RTCA DO-178B standard into a UML profile and to use them to facilitate the communication between different stakeholders in software development. One of the purposes of the profile, to make requirements more understandable to all stakeholders, is similar to that of our approach. However, we aim to do it by describing the occurrences of hazards to the control and safety software engineers and by enabling semi-formal specification of requirements with notations familiar to engineers and explicit traces between hazards and requirements, where as the concepts of the airworthiness profile are extended from the airworthiness standard.

8. Discussion and conclusions

It might be beneficial if both the basic control systems and the safety-related systems could be developed with similar or same tools. The industrial trend is moving towards model-based techniques also in development of safety-related systems. For example, IEC 61508 states that *automatic software generation* may aid completeness, correctness and freedom from intrinsic design faults in architecture design. As a consequence, the issue of how to develop safety-related systems with model-based techniques is important.

Unification of the development tools and notations could benefit both the development of basic control systems and safety-related systems. For example, developers would be capable of developing both kinds of systems and understand the couplings between the systems. Development of basic control systems could also benefit from the ability to generate documentation from design that would be necessary to support model-based development of safety-related systems. It could also lead to unification of basic control systems and safety systems into single systems.

In this paper, we have discussed the development of safety-related systems and applications from the points of view of both IEC 61508 standard and model-based development. In model-based development of safety-related applications, documentation is a focal asset that must be addressed. To achieve at least part of the goals of the standard including traceability, correctness, completeness and unambiguousness, we have studied modelling notations that could be used in modeling of hazards and requirement. In order to provide an exemplary model utilizing the concepts, the new modelling concepts were added to the UML AP Tool. Our approach to fulfil the mentioned goals of the standard with the modelling concepts and features is summarized in table 1.

Property	Supporting modelling features
Traceability	Traces and traceability matrices between modelling artefacts.
Correctness	Formal and semi-formal notations, visibility of hazards and requirements to developers.
Completeness	Traceability and model checks to make sure that requirements and hazards are not overlooked.
Unambiguousness	Semi-formal and user-familiar modelling notations.

Table 1. Summary of goal properties and supporting modeling features.

One issue in shifting towards model-based techniques is that in development of safety-related systems, tools should be either verified or proven in use. Verification of a complete tool set supporting model-based development would be a vast project. Collection of usage data, however, could be possible from the development of basic control systems that often include safety-function-like interlockings. From the point of view of the authors, the main difference between interlockings and safety functions is that interlockings are not certified. Consequently, they can be designed to be more complex. However, the basic needs may be very similar to those of safety functions.

The authors acknowledge that the proposed methodology and the modeling concepts still require further development and exemplary modeling cases with which they can be assessed and further developed. The development of safety-related applications could also benefit from simulation capabilities. The approach presented by the authors in [13] could be integrated to development and design-time verification of alternative approaches to achieve safety. With such a development approach, it could be possible to run test-like simulations to test the approach to achieve safety. In our future projects, we will collaborate with the industry in order to gain deeper insight about re-occurring safety needs.

References

[1] S. Friedenthal, A. Moore and R. Steiner. “A practical guide to SysML”. Morgan Kaufmann OMG Press, San Francisco. 2008

[2] D. Hästbacka, T. Vepsäläinen and S. Kuikka. “Model-driven development of industrial process control applications”. Journal of Systems and Software (2011), doi:10.1016/j.jss.2011.01.063

[3] T. Vepsäläinen, S. Sierla, J. Peltola and S. Kuikka. “Assessing the Industrial Applicability and Adoption Potential of the AUKOTON Model Driven Control Application Engineering Approach”, The Proceedings of the 8th International Conference on Industrial Informatics. Osaka, Japan, July 13-16, 2010.

[4] T. Ritala and S. Kuikka. ”UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry”, The Proceedings of the 5th IEEE International Conference on Industrial Informatics, Vienna, Austria, 23-27.7.2007, pp. 885-890.

[5] International Electrotechnical Commission, IEC 61508: functional safety of electrical/electronic/programmable electronic safety-related systems. parts 1-7. 2010

[6] J. Paalijärvi, M. Katara, M. Karaila and T. Parkkinen. “Agile development of safety-critical software for machinery: A view on the change management in IEC-61508-3”. The 6th International Conference on Safety of Industrial Automated Systems SIAS 2010, Tampere, Finland, 14-15 June, 2010

[7] C. Jones. “Software quality in 2008: A survey of the state of the art”. 2008. Software Productivity Research LLC. http://www.jasst.jp/archives/jasst08e/pdf/A1.pdf (achieved 13.2.2011). 59 p.

[8] B. Douglass. “Analyze system safety using UML within the Telelogic Rhapsody environment”. IBM Corporation, 2009.

[9] Object management Group, “UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification”. Version 1.1. OMG. 2008.

[10] M. de Miguel, J. Briones, J. Silva, and A. Alonso, “Integration of safety analysis in model-driven software development,” IET Software, vol. 2, no. 3, pp. 260–280, 2008.

[11] Object Management Group, “OMG Systems Modeling Language (OMG SysML™), Version 1.1, OMG 2008, online at: http://www.omg.org/spec/SysML/1.1.

[12] International Electrotechnical Commission, IEC 61804: Function blocks (FB) for process control – part 1: Overview of the system, 2003.

[13] T. Vepsäläinen and S. Kuikka. "Simulation Assisted, Model-Based Development of Safety Related Interlocks", to appear in 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications. Noordwijkerhout, Netherlands, July 29-31, 2011. (accepted)

[14] International Electrotechnical Commission, IEC 62424: Specification for Representation of process control engineering requests in P&I diagrams and for data exchange between P&ID tools and PCE-CAE, 2008.

[15] AutomationML, www.automationml.org

[16] R. Guillerm, H. Demmou and N. Sadou. “Information Model for Model Driven Safety Requirements Management of Complex Systems”. In: First International Conference on Complex System Design and Management, Paris, France, October 27-29, 2010.

[17] M. Biehl, C. DeJiu, and M. Törngren. “Integrating safety analysis into the model-based development toolchain of automotive embedded systems”. In: LCTES 2010, pp 125-132, New York, NY, USA, 2010. ACM.

[18] G. Zoughbi, L. Briand and Y. Labiche. “A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software”. In: MODELS 2007. LNCS, vol. 4735, pp. 574–588. Springer, Heidelberg 2007.

Publication 7

Vepsäläinen, T., Kuikka, S. (2014) Design Pattern Support for Model-Driven Development. Proceedings of the 9th International Conference on Software Engineering and Applications. Vienna, Austria, August 29-31, 2014, pp. 277-286.

DOI: 10.5220/0004990002770286

Design Pattern Support for Model-Driven Development

Timo Vepsäläinen and Seppo Kuikka
Department of Automation Science and Engineering, Tampere University of Technology, Tampere, Finland

Keywords: Design Pattern, Model-Driven Development, Tool Support.

Abstract: Design patterns document solutions to recurring design and development challenges. UML, as the de-facto modeling language in software development, aims to support defining and using patterns in models. However, as is demonstrated in the paper, the support is not sufficient for all kinds of patterns and all meaningful ways to use patterns. In this paper, the use of design patterns is suggested for documentation purposes in Model-Driven Development. The pattern support of UML is complemented with an approach that does not constrain the structures that can be used in patterns. The approach, which is tool supported in a model-driven development environment for control applications, also enables specification of part of the information content of patterns that UML leaves intact. The developed tool support includes instantiating and highlighting patterns in models and gathering of traceability information on use of patterns.

1 INTRODUCTION

Design patterns document proven solutions to challenges that keep arising in design and development work. Patterns capture expert solutions for reuse purposes for both expert developers and less experienced ones. In UML modeling, support for using patterns is only partially enabled by the language. The support for the use of patterns is based on Collaboration and CollaborationUse concepts (OMG, 2011) that have been developed along the entire language specification from parameterized collaborations (Sunyé et al., 2000).

However, in addition to the standard approach, many tool vendors, e.g. No Magic (No Magic, 2014), have implemented additional pattern support in a more ad hoc manner. Such support for patterns is in many tools based on informal UML templates that can be copied into design models to create instances of the patterns. In addition, copying the templates may utilize wizards that enable modifying pattern occurrences to specialized forms, by e.g. selecting existing elements to pattern-specific roles.

However, without referencing pattern definitions the information about the occurrences is endangered to vanish. With application specific names of e.g. properties, classes and interfaces, the occurrences are difficult to notice later for both developers and the tools. To avoid losing this information, patterns should be modeled and their occurrences marked in the models.

With its concepts, UML aims to support the definition of patterns in library models and their instances in models. It appears that the collaboration concepts of UML have been designed with traditional GoF (Gang of Four) (Gamma et al., 1994) patterns in mind: with focus on co-operating objects as properties of classes. However, as will be demonstrated, the UML concepts may not be sufficient for all kinds of patterns and foreseeable, meaningful ways to use patterns. Nevertheless, when patterns are utilized in software projects, documenting their use in models could be of great value. Especially this is the case with development processes that emphasize the use of models, e.g. Model-Driven Development (MDD).

In addition to solutions, design patterns include textual information about, for example, their contexts and the problems being solved. In (Alexander, 1979), the pattern concept is defined as a three-part rule expressing a relation between a context, a problem and a solution. A design pattern defined with the UML concepts, however, is likely to provide only information about the solution part of the pattern leaving the other important aspects unspecified.

This paper addresses the aforementioned issues. A pattern modeling approach is presented, which is less restrictive than that of UML and enables

Vepsäläinen T. and Kuikka S..
Design Pattern Support for Model-Driven Development.
DOI: 10.5220/0004990002770286
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 277-286
ISBN: 978-989-758-036-9

specification of part of the information content that UML does not address. The approach is tool supported in UML AP (UML Automation Profile) tool environment (Vepsäläinen et al., 2008) for MDD of control applications. The contributions of this paper are as follows. A set of concepts for defining and using design patterns is presented and rationalized. The benefits of the concepts are pointed out and compared to pattern support in UML. The use of patterns and pattern markings is proposed to benefit development work, documentation and learning of developers within MDD.

The rest of this paper is organized as follows. Section 2 reviews work related to modeling and facilitating the use of design patterns in UML context. Section 3 outlines and discusses how the use of patterns could benefit specifically MDD. The means of UML to define and use patterns are presented in section 4, in addition to pointing out shortcomings in the support with use of well-known example patterns. Section 5 presents a new approach to model patterns and pattern instances and illustrates the tool support developed based on the concepts. Before conclusions, section 6 discusses the work presented and future work to be done.

2 RELATED WORK

The roots of design patterns, as a concept, lie in building architecture and work of Alexander, see (Alexander et al., 1977) and (Alexander, 1979). In software development, the use of patterns began to gain popularity after publication of the Gang of Four (GoF) patterns (Gamma et al., 1994), in which the application area was object oriented programming and software, but not so much modeling. However, support for patterns was also developed to UML.

In addition to area of expertise, e.g. building and software engineering, design patterns vary in their abstractness and levels of details specified. For example, (Lasater, 2010) describes patterns as design tools to improve existing *code* whereas (Buschmann, 1999) focuses on *architectural* patterns that can have varying implementations. Patterns for safety systems development can be found e.g. in (Rauhamäki et al., 2013), the patterns mainly describing roles of elements.

The need for automated tool support to define and use design patterns in models has been identified by several researchers. Support has also been developed for specifying patterns, identifying pattern instances, detecting parts in models where patterns could be used as well as for instantiating and visualizing patterns.

(France et al., 2004) presents a formal pattern specification technique that is based on UML. It is intended for specifying design patterns and checking conformance of pattern instances to their specifications. In (France et al., 2003), automatic transformations are developed for refactoring patterns into models. The approach is based on specifications of pattern-specific problems, solutions and problem-to-solution transformations.

Detection of points in models where design patterns could be used has been studied, among others, in (Briand et al., 2006). In the paper, detection rules are specified with OCL (Object Constraint Language) and combined with decision trees. Detecting design pattern instances has been studied in (Tsantalis et al., 2006) the approach being based on representing both the models and patterns with graphs and applying graph similarity scoring.

Automating application and evolution of design patterns has been proposed and studied in (Dong and Yang, 2006), (Xue-Bin et al., 2007) and (Kajsa and Majtás, 2010). In (Dong and Yang, 2006), QVT (Query/View/Transformation) transformations are developed for evolving pattern applications to new ones, e.g. adding new observers to an Observer pattern instance. (Xue-Bin et al., 2007) uses XSLT (Extensible Stylesheet Language Transformations) for pattern-specific transformations to add patterns. The work in (Kajsa and Majtás, 2010) utilizes model transformations that are guided with UML stereotypes to mark the points to which the patterns should be added.

Visualizing design patterns in model diagrams has been addressed in (Dong, 2002) and (Jing et al., 2007). (Dong, 2002) presents several notations to highlight and distinguish patterns and pattern-related elements in diagrams. Among them is the collaboration notation that is also used in this paper. In (Jing et al., 2007), a UML profile is developed for specification of pattern roles that elements in pattern occurrences play. Based on the profile, the authors have developed a web service tool that integrates to e.g. Rational Rose to visualize patterns.

3 DESIGN PATTERNS TO FACILITATE MDD

Design patterns provide many general, well-known benefits to development work. For example, they encapsulate knowledge and experience, provide

common vocabulary for developers and enhance documentation of designs (Agerbo and Cornils, 1998).

More recently, design patterns have been seen to mark points in which developers have been potentially faced with challenges. Design patterns can be considered as predefined, reusable design decisions. However, they may require configurations for specific applications (Jansen and Bosch, 2005). Patterns are proven and general whereas design decisions are more tentative, specific to an application and also possible to be choices between solutions (Harrison et al., 2007), e.g. patterns. By marking a design pattern instance, a developer thus not only instantiates and configures a solution but marks a challenge and documents a decision.

The use of patterns in models can thus extend the documentation value of the models with architectural knowledge. However, especially patterns could be valuable in MDD in which the purpose is to shift development efforts from documents to models. To demonstrate this point, we discuss their use to a few selected purposes.

Patterns can be used to gather statistics. When patterns are marked in models that are used throughout the development process, it is possible to gather statistics on the use of the patterns. Pattern markings promote traceability between the solutions (of the patterns) and their use in software products. It is possible to study and compare work and preferred solutions of developers. Companies and teams can set up rules for using patterns in order to unify designs. For example, it could be agreed that a specific kind of challenge is always solved with a standard way in applications of a specific domain.

Also metrics could be defined to evaluate software products in an application domain or work of different developers. Extensibility and modifiability, for example, are quality attributes that many classic design patterns aim to improve. As a consequence, it is possible that similar software products could be compared in terms of preferred quality attributes by comparing the patterns and amount of patterns used in the products.

Design patterns can promote learning of new developers, too. When best practices and expert solutions are documented as patterns and pattern instances marked in design models, the models can be used as training material. New developers can look for pattern instances, in which kinds of contexts they have been used and how they have been used by experienced developers. Optimally, design pattern instances could be highlighted in models and diagrams in order to ensure their discovery. Diagrams with pattern annotations could also be used as parts of written documents when copied to such documents, when necessary.

It can be argued that the mentioned benefits are not restricted to the use of patterns in MDD only. However, the benefits from increasing the documentation value of *models* are of special importance in MDD. This is because one of the objectives of MDD is to gain benefits by changing the focus of development efforts from documents to models. If the aim is not to produce written documents in which challenges, decisions and solutions could be included, the only places where they can be added are the models.

On the other hand, in development practices other than MDD there may not always be need to model all parts of the developed systems. If all parts and aspects are not modeled, being able to produce e.g. statistics from models may not result in unbiased information on use of patterns. It is possible that the results from systematic use of patterns in models could be more usable in MDD context than with traditional development processes.

4 SUPPORT FOR DESIGN PATTERNS IN UML

In UML, patterns are defined with the Collaboration concept that extends both the StructuredClassifier and BehavioredClassifier concepts, similarly to the Class concept of the language. A pattern is a set of cooperating participants that are owned by a Collaboration instance as its properties, similarly to properties of a class. For each pattern-specific role there should be a property owned by the Collaboration. Required relationships between the participants are specified with connectors between the properties. The features required from the participants are defined by the classifiers (e.g. classes or interfaces) that are used as types of the properties.

Pattern instances are represented with the CollaborationUse concept. A CollaborationUse represents an application of a Collaboration (pattern) to a specific situation. CollaborationUses are owned by classes to contents of which the Collaborations (patterns) are applied. Contents (properties) of the applying classes are bound to roles (properties) of the Collaborations with Dependencies that are called role bindings. The entities (properties) playing the roles in the pattern instances must be owned by the classifiers owning the CollaborationUse elements.

Graphically, Collaborations and CollaborationUses can be defined in composite structure diagrams (CSDs). In case of defining a Collaboration (pattern) the root element of the diagram is the Collaboration, whereas in case of a CollaborationUse the class owning it. In other diagrams, CollaborationUses can be visible in compartments related to the applying classes, if supported by the tool being used.

4.1 Challenges with the UML Pattern Modeling Approach

The approach of UML for defining and using design patterns is formal and well-defined. However, when compared to, for example, literature presentations of many well-known patterns, the UML concepts cannot be used in a literature prescribed way. A CollaborationUse cannot be used e.g. in a class diagram describing classes of a package because in that case the participants would be classes (instead of properties) and owned by a package (instead of a class). For example a set of classes as in Figure 1 could not be marked as an Observer (Gamma et al. 1994) pattern instance.

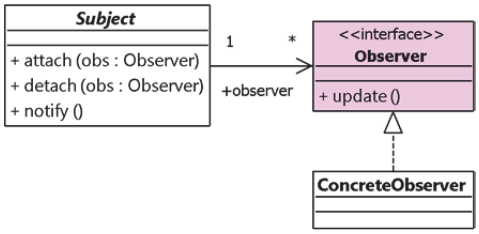

Figure 1: A class diagram illustrating the Observer pattern.

A rationale for claiming that the familiar structure in the figure cannot be an Observer instance could be that a class diagram does not yet indicate definite occurrence and use of instances of the classes in the pattern-specific way. Instead, the UML approach would be to define another class, create instances of the classes (of the figure) as properties of the other class and connect them to use the services of each other. Graphically this could be done with CSDs.

CSDs were not available at the time e.g. Observer pattern was authored, which is a possible explanation for the tool support to differ from the literature (or vice versa). However, from a pragmatic point of view, it may not be worthwhile to require definition of the class instances in CSDs because CSDs are not used as commonly (e.g. in industry) as class diagrams are. On the other hand, if a developer deliberately designs classes so that they can be used according to a pattern, it should be possible for her to mark the decision, e.g. for documentation purposes.

Another example related to the lack of pattern modeling capabilities in UML is related to architectural patterns. A well-known example of such a pattern is the Layers pattern (Buschmann, 1999). An intuitive means to illustrate the use of Layers in a UML model could be to present the packages and classes that an application is built of in a layered-like orientation as in Figure 2. One could also use component diagrams and arrange the components to a layered like orientation, like in (Buschmann, 1999) pp.35. However, neither of these approaches could be marked as a Layers instance. Packages, that class and component diagrams are used to describe, are not classes and thus cannot own CollaborationUses. And if they could, the packages and components would not be properties of a class.

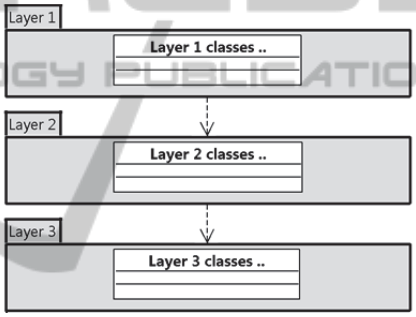

Figure 2: A layered architecture pattern illustration in a class diagram.

Observer and Layered Architecture patterns were used as examples above because of their familiarity. However, they are not the only patterns that may be difficult to apply in UML models. When patterns and pattern instances are defined and applied as contents of classifiers, use of patterns to describe aspects other than those related to classes and properties becomes difficult. Especially this can be seen to restrict the support for architectural patterns.

Related to pattern languages, UML does not define means to specify relations between patterns. According to the language specification (OMG, 2011), Collaborations can extend others. However, there is no means to specify, for example, that after applying a pattern it could be advisable to apply another, related pattern.

Lastly, the means of UML for defining information content of patterns other than solutions, e.g. context and problem, are limited. The Collaboration concept does not include textual or other kinds of properties for such purposes.

5 A NEW PATTERN MODELING APPROACH

Generally, the concepts that can be used in models conforming to a modeling language are defined in the metamodel of the language. The concepts available in UML models, for example, are defined in the UML metamodel (OMG, 2011) which in turn has been defined with use of Meta Object Facility (MOF). The metamodel of the new pattern modeling concepts, with relations to existing UML concepts, is presented in the next sub-section.

5.1 Metamodel for Defining, Marking and using Design Patterns

What pieces of information a pattern is obviously required to include are a name (identifier), problem (that the pattern solves), context (in which the pattern can be applied) and the solution, as also suggested in (Alexander, 1979). On the other hand, as argued in the previous section, the modeling approach should not restrict the nature of solutions in patterns. Practical patterns may consist of practically any modeling elements, e.g. components or class definitions. It should also be possible for other modeling elements than classes to contain elements that are parts of a pattern instance.

The basic concepts of the new pattern modeling approach are depicted in Figure 3 that has been divided into two parts. The concepts on the left-hand side are aimed for *defining* patterns whereas the concepts on the right-hand side for *using* and *marking* patterns instances. Although they are part of the same metamodel, it is assumed that design patterns could be defined in specific library models (preferably by experienced developers) and their instances used in application models (of the systems being modeled). Similar division of concepts exists already in UML related to profiles and stereotypes. Stereotypes are defined by experts in profiles and then used in a number of application models. Although stereotypes can be considered as tools for design work and altering the semantics of modeling elements, they are defined in UML models similarly to the concepts that they specialize.

The Pattern and PatternApplication concepts are aimed for defining patterns and pattern instances, respectively. Their UML counterparts are the Collaboration and CollaborationUse concepts. However, instead of defining (only) contents of a classifier, Patterns contain textual information which has been structured based on the canonical form of patterns (Appleton, 1997) with addition of Consequences from the Alexandrian form (Alexander et al., 1977).

The Pattern concept is extended from the UML PackageableElement concept so that Patterns can be defined within package hierarchies. The main contents of Patterns are PatternRoles that are used to specify structural and behavioral *roles* specific to the Patterns. Multiplicities define the limitations to numbers of modeling elements playing the roles in pattern instances. PatternRoles can also refer to template elements that are specific to the roles. Their purpose is to enable development of tool support to facilitate the creation of pattern instances.

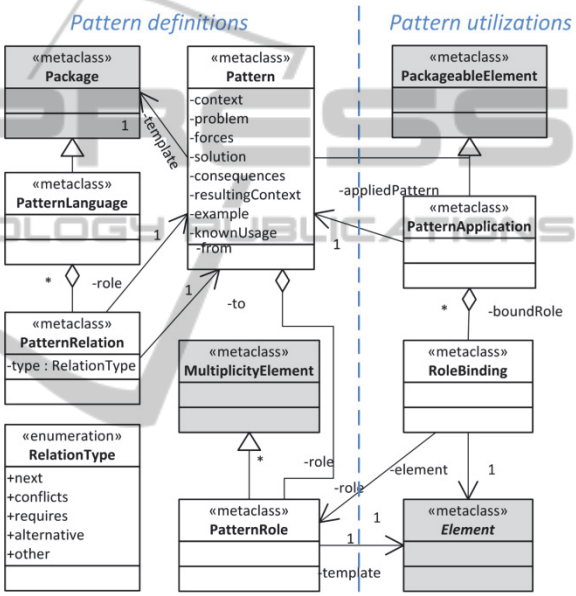

Figure 3: The metamodel of the new pattern modeling concepts; UML concepts are highlighted with grey color.

RoleBindings are owned by PatternApplications and they bind pattern instance specific elements to the roles of the patterns. The metaclasses of bound elements are not restricted since (concrete) elements of UML all extend the abstract Element concept that is used as the type of the meta-reference. The same applies to SysML and UML AP modeling elements in the supporting tool; they can be used in patterns and pattern instances as well.

PatternLanguage concept is a lightweight approach to pattern languages, allowing patterns to be organized into hierarchies. With PatternRelations, patterns can be organized into (pattern) sequences describing meaningful orders of using patterns, and sequences combined to simple languages. Relations also allow the specification of alternatives, patterns requiring other patterns and patterns that conflict

with each other. This aspect is yet to be defined in more detail.

The major differences of the approach in comparison to plain UML are as follows. The roles of patterns have been separated from their template elements in the template packages. Pattern definitions may contain textual information. The model elements playing the roles in patterns and their instances are not restricted to be instances of any specific UML (or e.g. SysML) metaclass. Lastly, PatternApplications are owned by packages that are used in models in any case.

The concepts relieve the restrictions of UML so that, for example, the patterns presented in section 4.1 could be marked as instances of suitable pattern definitions. Since elements playing roles in a pattern need not be properties, for example class definitions of Figure 1 - or some other variation of the pattern – could be marked as an Observer pattern instance. A structure like that could also be marked as a pattern instance regardless of whether the constructs would be defined in the same or different package. It would only affect to which package should own the PatternApplication element. Constructing patterns from classes, packages and components is also possible, which would enable marking the structure of Figure 2 as an instance of the Layers pattern.

As a downside, the approach is less formal than that of UML. Because of the freedom to define patterns to consist of any elements, it is more difficult to confirm correctness of pattern applications, for example. Since the approach does not restrict the elements that play roles in a pattern instance to be owned by a single model element, it is also possible for pattern instances to disperse to several places in models due to, for example, model refactoring. That is, although simple checks of consistency can be automated with e.g. the multiplicity restrictions more responsibility over correctness of pattern definitions and instances is left for developers in the approach.

Another restriction of the approach is related to the portability of it to other tools, which is caused by the metamodel additions that the approach requires. This aspect is discussed in more detail in section 6.

5.2 Illustrative Example

To demonstrate the use of the concepts, they are used in an example to define Observer pattern and to apply it to a model. The starting point in the example is a situation in which a PressureControl class would need to be made capable of receiving notifications of new (pressure) measurements from a PressureMeasurement class. A class diagram illustrating this starting point is shown in Figure 4.

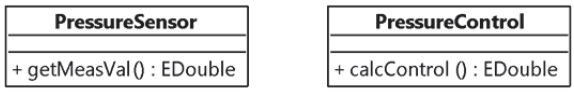

Figure 4: An example diagram before applying a pattern.

In order to apply Observer (Gamma et al., 1994), it needs to be first defined with the presented modeling concepts. A tree view of a model defining the pattern with the concepts is shown in Figure 5. The pattern is in the example defined in a Package that contains the Pattern element (Observer) as well as a template Package. The pattern includes roles related to it (Observer, Subject and ConcreteObserver). The classes and interfaces of the template package were illustrated in Figure 1; they also define several operations that are hidden from the figure below. Textual information related to the pattern, e.g. context and problem, is stored in the properties of the Pattern element.

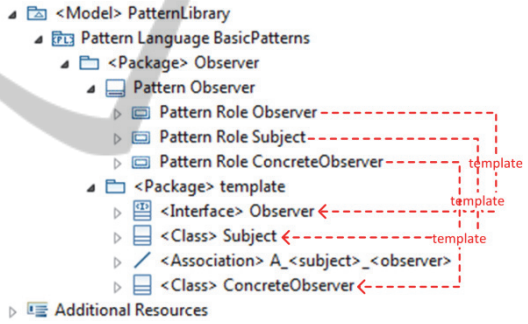

Figure 5: A tree view of Observer definition with the modeling concepts.

The example class diagram, after applying the pattern, is illustrated in figure 6. The diagram also illustrates how the pattern instance is visualized with the collaboration notation. The modifications from applying the pattern include addition of an interface (Observer), an interface realization as well as several operations specific to the role elements in the pattern, e.g. update(). These elements have been added based on the template elements illustrated in Figure 1.

Another view to the results is presented in figure 7 that illustrates the references between the model trees related to the pattern definition and pattern instance. The operations and other added model elements are contained in the model in a similar manner than any model elements. The information about the pattern instance, on the other hand, is

stored in a PatternApplication element. The PatternApplication contains the RoleBindings that link the pattern instance specific elements to the general roles of the pattern definition.

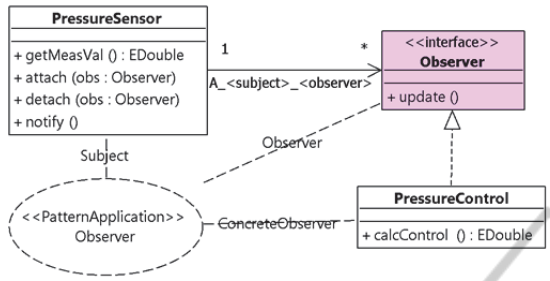

Figure 6: A visualization of an Observer pattern instance.

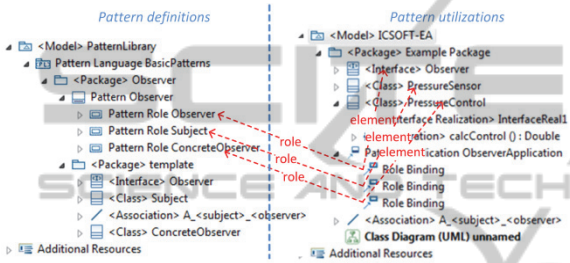

Figure 7: References from a pattern instance to definition.

5.3 Tool Support for using Patterns

With the tool support, the purpose has been to facilitate the use of patterns and to demonstrate the benefits from their use. The metamodel extensions to UML AP and UML modeling concepts, see Figure 3, were defined with Eclipse Modeling Framework (EMF) that is a Meta Object Facility (MOF) implementation used by the UML AP tool (Vepsäläinen et al., 2008). In addition to implementing the concepts, tool support has been developed to instantiate and to visualize patterns in models as well as to generate documentation from models. Of these functions, first two have been implemented with the core of the tool whereas the latter extends the documentation generation work in (Vepsäläinen and Kuikka, 2011).

5.3.1 Instantiating Patterns

Compared to instantiating patterns from templates in an ad hoc manner, the use of the presented concepts requires additional work. Defining patterns with the Pattern and PatternRole elements has to be done only once for each pattern. PatternApplications, however, need to be created and configured for each new instance. As such, it is natural that this task should be facilitated with tool support. In the tool, this task has been integrated to a wizard. Compared to existing pattern wizards in UML tools, the novelty of the wizard is in managing the new concepts.

The process of instantiating patterns is performed as follows. The user of the tool initiates the wizard from a tool menu. As a response, the tool scans through available pattern libraries in order to find available patterns. New libraries can be added to the tool by registering them with an (Eclipse) extension point developed for this purpose.

The user of the tool is provided with a list of available patterns. When selecting a pattern to apply, part of the textual information (problem, context and solution) related to the patterns is visible to the user, as illustrated in Figure 8. After selecting a pattern, the pattern (definition) that should be referenced by the PatternApplication to be created is known. In case of the design diagram root element being a package, the PatternApplication to be created can be owned by the package. Otherwise, it can be created to be owned by the package closest to the diagram root in the model hierarchy. The wizard proceeds to processing (iterating through) the pattern roles.

For each role, the wizard enables the user to select an existing element from the active diagram to act in the role. If the pattern in question defines a template, it is also possible to copy an element for the role from the template. For PatternRoles that the user has either selected an element for or copied it from the template, the wizard creates RoleBindings that bind the elements to the roles of the pattern. In case of using existing elements in roles of a pattern, their contents (elements owned by them) are compared and completed to correspond to those of the templates by copying missing contents.

Technically the wizard has been implemented so that it only collects the information from the user whereas actual model changes are performed at once after completing the wizard. The purpose of this is to enable possibility to collect model modifications to a single (undoable) command. However, currently undoing a pattern application requires manual work.

It is also possible to modify pattern instances after creating them. PatternApplications and RoleBindings can be selected from the outline view and modified with the properties view of the tool. Elements related to a pattern instance can also be re-organized and it is possible to apply more instances of compatible patterns. Information on which elements are part of a pattern instance is stored in a PatternApplication specific to the instance and the RoleBindings of it. They are not affected by additions of new elements or simple changes to the bound elements, e.g. re-naming or moving them.

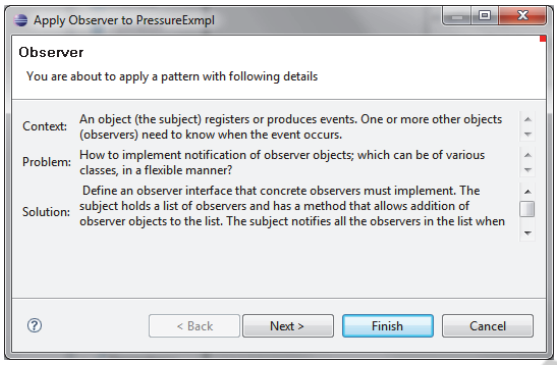

Figure 8: The pattern information page of the wizard.

5.3.2 Visualizing Patterns

Although pattern instances are always visible in the outline view of the tool, they are not visible in diagrams by default. This is rational since the amount of details in a diagram should be relatively small to keep it understandable. Patterns can also be considered as explanatory information that may not be required all the time. However, when pattern applications are necessary to be shown, e.g. for documentation or teaching purposes, it should be possible to visualize them in diagrams.

Visualization of a pattern is initialized from a menu of the outline view of the tool while at the same time selecting the PatternApplication to be shown. As a response, a dotted ellipse shape with lines to the model elements playing the roles in the pattern instance is created. The ellipse represents a PatternApplication (pattern instance) and contains the name of the pattern (definition). Connections to the role elements show the names of the corresponding pattern roles.

The graphical presentation of pattern instances is similar to CollaborationUses in CSDs, with addition of <<PatternApplication>> to distinguish between them. An example graphical presentation of an Observer pattern application was presented in Figure 6. In the figure the pattern has been applied to a client application model so that the names of the concrete classes are different from the names of the template classes, which were shown in Figure 1.

5.3.3 Patterns as a Part of Documentation

One of the main motivations of this work has been to use patterns for documentation purposes in MDD. Since design patterns and design pattern instances are modeled with dedicated elements, it is possible to track the design patterns that are used in a model of an application as well as the number of instances of the patterns. Since PatternApplications are owned by packages, it is possible to trace the parts of models in which a design pattern is used. Starting from packages, it is again possible to track the patterns that are used in the packages.

Exporting documentation is initiated by the user of the tool that selects the root of the model from the outline view, selects export functionality and then traceability information. First sheets of the generated (Microsoft Excel) spreadsheet are described in (Vepsäläinen and Kuikka, 2011) whereas last two are dedicated to design patterns.

The first of the new sheets lists the design patterns that are used in a model. The sheet is collected by searching all PatternApplication instances in the model. The number of instances for each design pattern (definition) as well as the total amount of patterns are calculated and shown. With traceability matrices, the sheet presents package to design pattern traceability (the patterns that are used in each package), design pattern to package traceability (in which packages each design pattern is used) and lastly design pattern to element traceability. In the latter matrix, each design pattern instance is traced to all elements that play roles in the instance. An example sheet presenting traceability for the pressure sensor example of Figure 6 is presented in Figure 9.

	A	B
1	**Design pattern usage in ICSOFT-EA package/model.**	
2		
3	**Statistics on design pattern applications:**	
4	**Design Pattern**	**Pattern applications**
5	Observer	1
6	**Total number of patterns**	1
7	**Total number of pattern instances**	1
8		
9	**Package --> design pattern traceability:**	
10	**Package -->**	**Applied pattern(s)**
11	Example Package	Observer
12		
13	**Design pattern --> package traceability:**	
14	**Design Pattern -->**	**Applying package(s)**
15	Observer	Example Package
16		
17	**Design pattern --> Element traceability:**	
18	**Design Pattern (Package) -->**	**Element with a role (role)**
19	Observer (Example Package)	
20		PressureSensor (Subject)
21		PressureControl (ConcreteObserver)
22		Observer (Observer)

Figure 9: An exemplary automatically generated traceability sheet.

The second of the new sheets focuses on design patterns themselves. At the beginning of the sheet a list of patterns, instances of which can be found from the model, is repeated with the amount of pattern instances. After this table, the sheet presents printouts of information for each design pattern used in the model including context, problem, forces, solution (textually), consequences, resulting context, example, and known usage.

6 DISCUSSION AND FUTURE WORK

This paper has discussed the use of design patterns in UML based modeling and their potential benefits in model-driven development. Shortcomings in UML design pattern support have been pointed out and an additional set of modeling concepts has been presented.

The need for a new approach to utilize patterns in models originates from the UML pattern modeling concepts that restrict patterns to describe contents of classifiers. The information content of actual published patterns, however, is not restricted to such a narrow scope. Patterns may not always concern concrete programming language level aspects and their information content is not restricted to solutions only. For example, solutions of patterns may consist of packages, components or even use cases. In addition, patterns include information about their contexts and problems for which the patterns provide the solutions.

The presented, simple set of modeling concepts enhances the UML limitations by enabling patterns to include textual information and to consist of practically any elements that a pattern author finds useful. As a downside, the approach leaves more responsibility over the correctness of patterns and pattern applications to developers. The portability of the approach to other tools is also questionable, which is caused by metamodel modifications.

The approach introduces new metaclasses to the MOF based UML metamodel so that implementing the approach in other tools would require similar additions. The other extension mechanism of UML, light weight profiles that consist of stereotypes, however, would not have enabled all the required additions. According to the UML specification (OMG, 2011), stereotypes cannot be used to insert new metaclasses or metareferences between existing metaclasses, for example. With stereotypes (without new metaclasses), it would have been possible to include the textual information in the Collaboration concepts of UML. However, CollaborationUses would still be owned by classes and their other specified constraints would still apply.

In future work, it is our intention to focus on safety related patterns, examples of which can be found e.g. in (Rauhamäki et al., 2013). Safety related systems constitute an application domain in which documentation is of special importance. This is because of the need to justify the safety of the developed applications against safety standards. For software safety functions, the standards focus on development methods, practices and solutions that are recommended for different levels of safety. On the other hand, safety standards require traceability between requirements, design, implementations and test cases, among others. This is the problem domain that we foresee to be possible to facilitate with safety pattern modeling and extending the presented documentation generation work.

7 CONCLUSIONS

Design patterns document solutions and capture expert knowledge to recurring challenges in design and development work. The scope of design patterns that can be found from literature varies in terms of area of expertise and abstraction level. Many patterns present rather conceptual solutions than solutions that could be copied or modeled always in the same way. However, although the UML concepts have been enriched along the development of the entire language, the pattern support is still restricted to collaborating properties of classes.

In this work, the issue has been addressed by defining and implementing a set of pattern modeling concepts that can be used to complement the UML concepts. The approach is not restricted to modeling of classifiers only but enables patterns to consist of practically any modeling elements that an author of a pattern finds useful.

Tool support for automating the use of the new concepts has been developed for instantiating patterns, visualizing patterns in diagrams as well as collecting documentation and statistics from models. The tool and concepts have been used by researchers working in the project. They have been found useful and will be used to gather more use experience in software engineering courses at the department of Automation Science and Engineering at Tampere University of Technology.

The tool supported functionalities are also related to the way in which design patterns could be used to facilitate model-driven development. Patterns enable including additional documentation to models. Patterns enrich models with information on challenges, points of decisions as well as traceability between solutions and their use in specific applications. Visualizing patterns in diagrams may both support learning of developers and increase the value of diagrams in written documents. Knowledge on pattern use can be gathered to statistics to compare applications and work of developers. Patterns and rules for using them can also be used to unify work of developers in teams and companies.

REFERENCES

Agerbo, E., Cornils, A. 1998, How to preserve the benefits of design patterns, ACM SIGPLAN Notices, ACM, pp. 134-143.

Alexander, C. 1979, The timeless way of building.

Alexander, C., Ishikawa, S., Silverstein, M. 1977, Pattern languages, Center for Environmental Structure, vol. 2.

Appleton, B. 1997, Patterns and software: Essential concepts and terminology, Object Magazine Online, vol. 3, no. 5, pp. 20-25.

Briand, L.C., Labiche, Y., Sauve, A. 2006, Guiding the application of design patterns based on uml models, Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on, IEEE.

Buschmann, F. 1999, Pattern oriented software architecture: a system of patters, Ashish Raut.

Dong, J. 2002, UML extensions for design pattern compositions, Journal of object technology, vol. 1, no. 5, pp. 151-163.

Dong, J., Yang, S. 2006, QVT based model transformation for design pattern evolutions, in: Proceedings of the 10th IASTED international conference on Internet and multimedia systems and applications.

France, R.B., Kim, D., Ghosh, S., Song, E. 2004, A UML-based pattern specification technique, Software Engineering, IEEE Transactions on, vol. 30, no. 3, pp. 193-206.

France, R., Chosh, S., Song, E., Kim, D. 2003, A metamodeling approach to pattern-based model refactoring, Software, IEEE, vol. 20, no. 5, pp. 52-58.

Gamma, E., Helm, R., Johnson, R.,Vlissides, J. 1994, Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education.

Harrison, N.B., Avgeriou, P., Zdlin, U. 2007, Using patterns to capture architectural decisions, Software, IEEE, vol. 24, no. 4, pp. 38-45.

Jansen, A., Bosch, J. 2005, Software architecture as a set of architectural design decisions, Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference onIEEE, pp. 109.

Jing, D., Sheng, Y., Kang, Z. 2007, Visualizing design patterns in their applications and compositions, Software Engineering, IEEE Transactions on, vol. 33, no. 7, pp. 433-453.

Kajsa, P., Majtás, L. 2010, Design patterns instantiation based on semantics and model transformations, in SOFSEM 2010: Theory and Practice of Computer Science, Springer, pp. 540-551.

Lasater, C.G. 2010, Design patterns, Jones & Bartlett Publishers.

No Magic, Inc. 2014, MagicDraw. Available: http://www.nomagic.com/products/magicdraw.html [2014, 1/23].

OMG, 2011. Unified Modeling Language Specification 2.4.1: SuperStructure, Object Management Group.

Rauhamäki, J., Vepsäläinen, T., Kuikka, S. 2013, Patterns for safety and control system cooperation, Proceedings of VikingPLoP 2013 Conference.

Sunyé, G., Le Guennec, A., Jézéquel, J. 2000, Design patterns application in UML, in ECOOP 2000—Object-Oriented Programming Springer, pp. 44-62.

Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., Halkidis, S.T. 2006, Design pattern detection using similarity scoring, Software Engineering, IEEE Transactions on, vol. 32, no. 11, pp. 896-909.

Vepsäläinen, T., Hästbacka, D., Kuikka, S. 2008, Tool Support for the UML Automation Profile - For Domain-Specific Software Development in Manufacturing, Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on.

Vepsäläinen, T., Kuikka, S. 2011, Towards model-based development of safety-related control applications, Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on.

Xue-Bin, W., Quan-Yuan, W., Huai-Min, W., Dian-Xi, S. 2007, Research and implementation of design pattern-oriented model transformation, Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on, IEEE.

Publication 8

Vepsäläinen, T., Kuikka, S. (2014) Safety Patterns in Model-Driven Development. Proceedings of the 9th International Conference on Software Engineering Advances. Nice, France, October 12-16. 2014, pp. 233-239. Available in ThinkMind: http://www.thinkmind.org/download.php?articleid=icsea_2014_9_40_10247

Safety Patterns in Model-Driven Development

Timo Vepsäläinen, Seppo Kuikka
Tampere University of Technology
Dept. of Automation Science and Engineering
Tampere, Finland
{timo.vepsalainen, seppo.kuikka}@tut.fi

Abstract— **Design patterns capture named solutions to recurring challenges in development work. With an appropriate, non-restrictive tool support, design patterns could also improve the documentation value of models in model-driven development. This paper extends the design pattern modeling approach of UML Automation Profile with safety-related information and suggests the use of patterns in models to document safety aspects. The modeling concepts are tool supported. In the paper, the concepts are used for exporting safety-related documentation. The documentation can be used to guide the selection of development techniques as well as to perform consistency checks with respect to safety integrity levels that are required from modeled applications.**

Keywords-Model-Driven Development; Design Pattern; Safety.

I. INTRODUCTION

Design patterns document solutions to recurring challenges in design and development work. As a concept, design pattern was introduced in the work of Alexander [1][2] related to building architectures. In software development, design patterns began to gain popularity after the publication of the Gang of Four (GoF) patterns [3] that were targeted to object oriented software engineering. Support for the use of patterns was also developed to Unified Modeling Language (UML). Today, UML is the de-facto software modeling language. With domain specific profiles, UML is also the modeling basis of many Model-Driven Development (MDD) approaches. However, the support for design patterns in UML is still focused on describing contents of UML Classes.

The idea of MDD is to use models as the primary engineering artefacts during the development of software systems. Models describe the systems and applications from different points of view and on different abstraction levels. In MDD, the development often starts from high abstraction level models, e.g., Computation Independent Models (CIM) as in Model Driven Architecture (MDA) [4]. Model transformations are used between the models to ensure their consistency and to produce refined models based on the earlier ones. Models also document the developed systems. However, in specific application domains the required information content of documentation is governed by regulations and standards, in addition to development needs.

Safety-related systems and applications constitute such a domain. The development process of safety applications as well as solutions and techniques to be used during the process is governed by standards, e.g., IEC 61508 [5]. In addition to using standard-compliant techniques, a developer of such a system must be able to prove the compliance of it. This is where the relevant documentation is needed.

The use of MDD to safety system development has been suggested by few researchers and even less MDD has been taken to industrial practice. The reason is not that safety standards would not allow the use of MDD techniques. Instead, for example "automatic software generation" is recommended as an architecture design technique by IEC 61508 [5]. Possible explanations for the scarce use of MDD techniques in the application area are, however, the strict documentation requirements. It is possible that given the strict requirements, MDD has not been seen to offer possibilities to improve the efficiency of the development.

The purpose of this paper is to extend a design pattern modeling approach of UML Automation Profile (UML AP) [6] to safety patterns. Safety patterns are design patterns that are applicable for safety-related systems and include additional information related to safety. They can be used by exporting documentation from models of the developed systems in which the patterns are used. The documentation generation is intended to facilitate development work by: 1) supporting traceability between applicable safety solutions and their use in systems, 2) enabling verification of safety levels of patterns in comparison to required safety levels and 3) guiding the selections of techniques and solutions.

The rest of this paper is organized as follows. Section 2 reviews work related to design patterns and use of design patterns in models and model-driven development. Section 3 recapitulates the recent pattern-related work that is extended in the paper. Sections 4 and 5 present the safety-related extensions to the pattern concepts and the developed tool support, respectively. Before conclusions, Section 6 discusses the work and the relevance of safety aspects in control system development in general.

II. RELATED WORK

Support for using design patterns in UML models is in the language based on Collaboration and CollaborationUse [7] concepts that are suitable for presenting patterns inside

 ISBN: 978-1-61208-367-4

UML Classes. The concepts have been developed along the language itself from parameterized collaborations that were utilized in, e.g., [8]. In addition to the standard approach, however, many tool vendors have developed additional pattern support in a more ad hoc manner. For example, MagicDraw [9] enables the specification of model element templates and copying the templates to models to instantiate patterns. Without pointing out pattern instances, however, the information on the occurrences is endangered to vanish.

To enable precise but practical use of patterns in UML, France et al. [10] have developed a pattern modelling approach using UML. Precise specification of pattern solutions is seen to enable tool support for building solutions from pattern specifications and for verification of the presence of patterns in design. In the approach, an overall pattern specification consists of a structural pattern specification specifying the class diagram view of the solution, and a set of interaction pattern specifications that specify the interactions in the pattern solutions.

Approaches to apply and evolve design patterns to UML models have also been developed with use of model transformations [11][12][13][14] using Query/View/Transformation (QVT) and Extensible Stylesheet Language Transformations (XSLT) techniques. Detection of design patterns in models, on the other hand, has been studied for example with use difference calculation [15], graph matching [16], graph similarity scoring [17], as well as graph decomposition and graph isomorphism [18].

In the approach of the authors, the novelty is neither in the approach to transform patterns into design nor in detecting pattern instances. Instead, a starting point in the work is that uses of patterns are design decisions that should be deliberately documented by marking the patterns. On the other hand, attention is paid to the questions how the pattern markings could be used to produce documentation in general and in safety-related application development in particular.

For safety-related systems, design patterns have been specified, for example, related to redundancy. In [19], Douglass presents 4 patterns to implement redundancy or redundancy-like behavior so that a task is performed in different channels or that another computing channel is used to observe the behavior of the main channel. Also IEC 61508 [5] in the 6th part of it presents several M out of N solutions in which the idea is to perform a calculation redundantly and to use voting to acquire a reliable result for it.

In the tables of recommended techniques and measures for software architecture design (annex A), IEC 61508 [5] also refers to a wide range of solutions that already have corresponding patterns in pattern literature. For example, the standards suggest the use of (different kinds of) redundancy [19], backward recovery (from faults) [20][21] and cyclic program execution [19]. Another example on use of patterns in the domain is related to documenting recurring arguments of safety cases in order to systematically collect and gain benefit from arguments of previous projects [22].

MDD of safety systems has been studied in the DECOS project [23] that is targeted to development of both critical and non-critical functions of embedded control systems. In the approach, the preferred means for specifying application functionality is Safety-Critical Application Development Environment (SCADE) which is based on formally defined data flow notation and enables simulation at model level and code generation.

UML based modelling and development of safety applications has also been facilitated with UML profiling techniques. In [24] the approach is based on extracting key concepts of a safety standard, RTCA DO-178B, to stereotypes with which it is possible for software developers to include safety-related concepts and properties in models. It can be assumed that such models suit well also for the purpose of producing documentation. However, we regard the work presented in this paper as an important complement to the approach. Whereas UML stereotypes are applied to single modelling elements, with patterns it is possible to link several elements in designs to patterns and roles of them. This is needed in order to characterize how a number of elements are used together to perform a task.

III. NEED FOR PATTERNS IN MDD

The key concept of MDD is to shift the development efforts from written documents to models that are used throughout the development process. For special purposes, e.g., safety system development, it could be possible to maintain separate documents. However, that would require additional work and could significantly reduce the potential to benefit from MDD. In a sense, it would also be against the central idea in MDD. A more appropriate approach would be to include the documentation in the models, in the first place.

A possible challenge in this objective is that models, in general, tend to be more applicable for representing solutions than rationale behind them. For example, many of the basic concepts of UML are similar to concepts of object oriented programming languages. UML models can be well used to answer the question how to implement, e.g., a class or a program. In the MDD context, it is even possible to generate code from models to avoid the manual programming work. However, information on why something has been designed in the way it has, is often missing. This information could be crucially important for, e.g., quality assurance and maintenance purposes.

Design patterns are a possible solution to improve the situation. Patterns document named, proven solutions that are well-known among developers and suited for solving recurring challenges and tasks. They are structured so that they consist of named parts that have responsibilities in the solutions. The solutions that patterns include may have crucial advantages. The use of design patterns and pattern instances in MDD and models could thus increase the value of models significantly. Patterns could 1) indicate the use of standard solutions in systems and specifications, 2) mark potential challenges (that are treated with the patterns), 3) make design more understandable (because of the use of the known solutions) and 4) clarify the roles of model elements in design, just to name a few benefits. In specific application areas, e.g., safety system development, the use of patters could even automate tasks and checks that are currently performed manually.

 ISBN: 978-1-61208-367-4

A. Design Patterns in UML

In UML, pattern definitions and pattern instances are defined with the Collaboration and CollaborationUse concepts of the language, respectively. Similarly to the Class concept, Collaboration extends the StructuredClassifier and BehavioredClassifier concepts. A pattern definition is in the language a set of cooperating participants that are Properties of a Collaboration. In a similar manner Properties can be owned by Classes. The features that are required from the participants are defined by the Classifiers that are used as types of the Properties. Graphically Collaborations can be presented in composite structure diagrams in which participants of a pattern are connected with Connectors.

A CollaborationUse represents an application of a pattern to another Classifier (Class). The CollaborationUse must be owned by the Class to the contents of which it (the pattern) is applied. Properties of the applying Class can be bound to the roles of the Collaboration with Dependencies. The entities playing the roles must be owned by the same Class instance that owns the CollaborationUse. In short, with the UML pattern concepts, patterns are seen to describe contents of Classifiers.

Pattern literature of today, however, is not restricted to contents of UML Classifiers only. For example, many well-known patterns such as the Layers pattern [25] (and many other architectural patterns) are intended to clarify the division of systems to, e.g., Components or Packages. However, marking the occurrence of such patterns may not be possible with the UML concepts. This is because Packages are not Properties or necessarily owned by Classes. With application domain specific extensions, the support for patterns in UML becomes even more constraining. In order to benefit from the use of patterns in MDD, a new approach to define and mark patterns in models is required. The approach should restrict neither the types of elements that play roles in patterns nor the types of elements to contents of which patterns can be applied.

B. The New Pattern Approach

The developed pattern modelling approach [6] uses a set of concepts that have been developed for defining patterns and marking pattern instances in models. In the approach, pattern instances are not owned by Classes but Packages that are used in models in any case. The elements playing pattern specific roles in pattern instances can be any direct or indirect contents of the Packages and instances of any metaclass, instead of Properties only. Pattern definitions include textual properties that are essential information content in patterns. Lastly, the element roles in pattern definitions are separated from the template elements that are used in automating the application of patterns.

The approach is tool-supported including functions for instantiating patterns, exporting statistics and traceability information related to the use of patterns as well as for visualizing patterns in diagrams [6]. Patterns are instantiated to models with the use of a wizard that performs pattern specific modifications to the models, according to user selections. Markings of pattern instances are also created automatically by the wizard.

Statistics and traceability information on patterns can be exported to MS Excel files. Statistics include lists of design patterns that are used in a model including the number of instances for each pattern. Patterns are traced to Packages with traceability matrices to indicate the patterns that are used in each Package and vice versa. Visualizing patterns in diagrams utilizes the Collaboration notation of UML and presents pattern instances with dotted ellipses. Model elements that play pattern specific roles in the instances are connected to the ellipses with dotted lines. The tool support for the use of patterns can be used in any UML, Systems Modeling Language (SysML) or UML Automation Profile (AP) models and diagrams in UML AP research tool [26].

IV. SAFETY PATTERN METAMODEL

With extensions to safety aspects, the purpose has been to experiment how design patterns could specifically support documentation of safety applications. Most importantly, the extensions to the pattern modeling concepts, see Figure 1, include a specific SafetyPattern. SafetyPatterns are design patterns that have been identified to be related to safety. To distinguish the concepts that are used for defining patterns from those used to mark pattern instances, the Figure has been divided to two parts. The new (in comparison to [6]) concepts are in the Figure high-lighted with grey color.

A SafetyPattern is, thus, a design pattern that has been identified to be related to safety and that may have recommendations for applications of different *safety levels*. With safety systems, we refer to systems that perform *safety functions* the correct operation of which is required to ensure the safety of a controlled process. The safety levels in the metamodel correspond to the 4 Safety Integrity Levels (SILs) in IEC 61508 [5]. In general, a SIL determines the probability of correct functioning of a safety function, SIL1 being the lowest and SIL4 being the highest level. For traditional, e.g., electrical safety systems it is possible to determine SILs statistically. However, due to the systematic (vs. random) nature of software faults, the statistics approach cannot be applied to software. For new software components there would not even be statistics available. In IEC 61508, this problem is solved by focusing on development techniques and solutions the use of which are documented. For each SIL and for each development phase, the standard specifies a set of techniques that can be highly recommended (HR), recommended (R) or non-recommended (NR) or with non-specified recommendation (NS). The alternatives in the Recommendation (enumeration) in the metamodel correspond to these alternatives.

The purpose of the SafetyCatalogue concept is to collect together (from various pattern sources) related SafetyPatterns. Catalogues contain patterns that should be used together and to which sets of patterns that are used in models can be compared. Patterns in a catalogue can be related to, e.g., a phase in development or a specific purpose. For example, IEC 61508 [5] includes lists of techniques to be used during specific software development phases. For software architecture design, for instance, the standard mentions 27 techniques and/or measures, some of which are non-recommended or alternatives to each other.

 ISBN: 978-1-61208-367-4

Relations between Patterns can be modeled with the PatternRelation concept that has been extended with a Specialization relation. The background of the new (specialization) relation is an observation that many solutions (such as redundancy) that are recommended by safety standards actually have families of related, specialized pattern versions in pattern literature. With the Specialization relation, the purpose is to enable the use of general SafetyPatterns in SafetyCatalogues but in such a way that patterns specializing the general patterns can be considered as their alternatives.

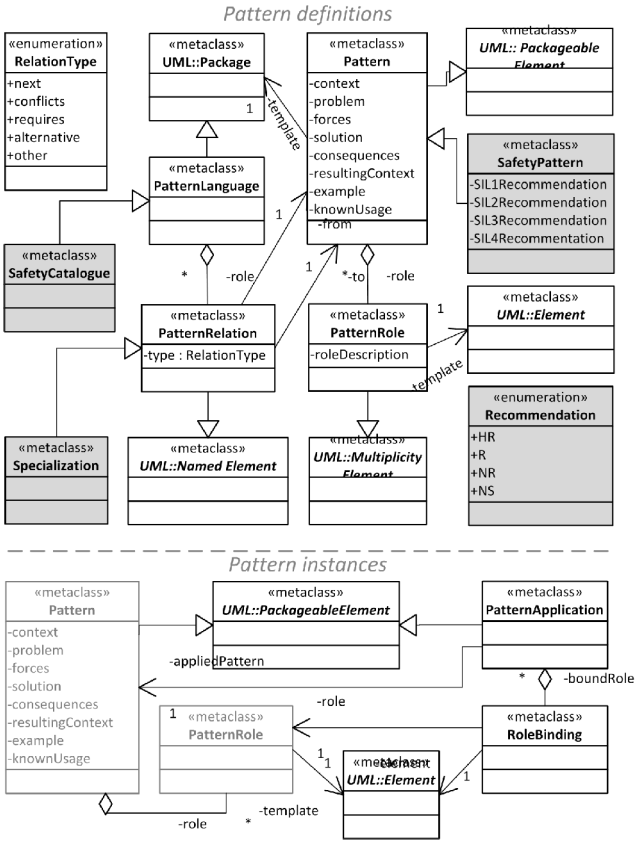

Figure 1. The new concepts for defining and using safety patterns.

The modeling concepts have been implemented to UML AP Tool [26]. With the implementation, the purpose has been to demonstrate how the concepts can be used to generate safety-related documentation. The implementation of the concepts uses Eclipse Modeling Framework (EMF) as a meta-modeling framework, with which the new concepts have been defined by extending the existing UML AP modeling concepts. The developed documentation generation extends the work presented in [6] and [27] that already addresses, e.g., traceability of requirements.

V. FOR GUIDANCE AND DOCUMENTATION

In this Section, we present three example documentation sheets. The generation of the sheets has been automated with use of the concepts. In addition to discussing how the sheets can be used, the following sub-Sections will briefly describe how the sheets are compiled from models.

The first of the sheets to be presented was created based on a SafetyCatalogue that had been defined to correspond to recommendations of IEC 61508 to software architecture design. The latter two example sheets compare a set of SafetyPatterns that is used in an example model to another SafetyCatalogue. The generation of the sheets relies on patterns that have been identified to be related to safety and that include recommendations for the different levels of safety.

A. Safety Catalogue Sheet

The purpose of the Safety Catalogue sheet is to enable illustrating SafetyCatalogues in a tabular form that is similar to the form of recommendation tables of IEC 61508 [5] (annex A of part 3 of the standard). On one hand, the sheet has been developed to facilitate the development of SafetyCatalogues, including checks of their conformance to standards. The tabular presentation can be used also during development to look for possible patterns or solutions that should be applied during specific design phases.

In addition to recommendations of safety standards, the sheet enables illustrating custom catalogues of SafetyPatterns for which there may not be standard recommendations. Nevertheless, such patterns may provide solutions to similar problems and be alternatives to each other. On the other hand, it may be meaningful to represent in which order such patterns should be applied so that composing pattern catalogues with next and alternative relations can be useful.

The Safety Catalogue sheet is compiled as follows. PatternApplications of an exported model are iterated through to find all SafetyPatterns that are used in the model. The SafetyPatterns are iterated through to find the SafetyCatalogues in which they appear. The list of the catalogues is provided to the user of the tool. The selected catalogues are printed to separate tables starting from their first patterns that are assigned number 1 in the tables. Next and alternative SafetyPatterns can be found with use of the PatternRelations. Alternatives are in the tables assigned same numbers but different letters, to indicate them being alternatives to each other. Recommendations of the SafetyPatterns to SILs are printed to the tables.

Safety Catalogue: "IEC 61508 Architecture Design"

#	Pattern	SIL 1	SIL 2	SIL 3	SIL 4
1	Fault detection	NS	R	HR	HR
2	Error detecting codes	R	R	R	HR
3a	Failure assertion programming	R	R	R	HR
3b	Diverse monitor techniques (with independence)	NS	R	R	NS
3c	Diverse monitor techniques (with separation)	NS	R	R	HR
3d	Diverse redundancy	NS	NS	NS	R
3e	Functionally diverse redundancy	NS	NS	R	HR
3f	Backward recovery	R	R	NS	NR
3g	Stateless software design	NS	NS	R	HR
4a	Re-try fault recovery mechanism	R	R	NS	NS
4b	Graceful degradation	R	R	HR	HR
5	Artificial intelligence - fault correction	NS	NR	NR	NR
6	Dynamic reconfiguration	NS	NR	NR	NR
7	Modular approach	HR	HR	HR	HR
8	Use of trusted/verified software elements (if available)	R	HR	HR	HR
...					

Figure 2. An example generated Safety Catalogue sheet.

 ISBN: 978-1-61208-367-4

An example Safety Catalogue sheet can be seen in Figure 2 that presents a part of a printout of a catalogue of techniques or measures that IEC 61508 recommends for software architecture design. In the table, patterns can be highly recommended (HR), recommended (R) or non-recommended (NR) or with non-specified recommendation (NS). To avoid repeating a table of the standard, the table includes only 15 techniques that have been modeled as patterns. By looking at the table, however, it also becomes clear that pattern literature already includes specialized versions of many of the techniques, for example to implement redundancy [19].

B. *Safety Catalogue Conformability Sheet*

Whereas the purpose of the Safety Catalogue sheet is to enable presenting catalogues of SafetyPatterns, the purpose of Safety Catalogue Conformability sheets is to present how a set of SafetyPatterns (that are used in a model) conforms to a SafetyCatalogue. Similarly to the previous sheet, the conformability sheet serves both the guidance and documentation purposes. In addition, the table presents to which SILs the set of SafetyPatterns would be applicable.

The sheet is compiled as follows. In a similar manner than in the case of the previous sheet, the SafetyCatalogues related to the model are collected to a list from which the user may select the desired ones. General structure of the sheet is similar to the previous sheet. However, the SafetyPatterns of the catalogue that are used in the model are indicated with light grey color. In addition, the table presents whether the set of (used) patterns is compatible with each SIL. Compatibility of the used patterns is illustrated with green color and incompatibility with red color. Incompatibility can result from both using a non-recommended pattern or not-using a recommended (or highly recommended) technique or any of its alternatives.

The last two rows of the table also present the numbers of patterns (excluding alternatives) that would be recommended for each SIL and how many of them have been actually applied. As such, the table also answers the question how many techniques (more) should be applied in order to conform to the catalogue for each SIL.

Conformability to "IEC 61508 Software safety requirements specification"

#	Pattern	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	R	R	HR	HR
1b	Formal methods	NS	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR
	Pattern usage:	4/4	4/4	4/4	4/4
	Pattern usage (%):	100.0	100.0	100.0	100.0

Figure 3. An example generated Safety Catalogue Conformability sheet.

An example Safety Catalogue Conformability sheet can be found in Figure 3. It presents the conformability of SafetyPatterns used in an example model to the software safety requirement specification techniques of IEC 61508 [5] that have been modeled as a SafetyCatalogue. According to the table (grey highlighting), it can be seen that a semi-formal modeling technique has been used, the software safety requirements specification supports both backward and forward traceability and that computer-aided specification tools have been used. The table also illustrates (with green color) that these choices are applicable to all SILs. In addition to the techniques used, it is not necessary to use any other technique (for requirements specification).

C. *Safety Pattern Traceability Sheet*

While patterns can have recommendations for different levels of safety, it is also possible to check their conformance to safety levels required from the safety functions. The purpose of the safety pattern traceability sheet is to trace safety requirements (of UML AP) to Packages that contain implementing design elements for the requirements and to SafetyPatterns that are used in the Packages. In addition to traceability, the table presents the safety levels (SIL) related to the requirements, Packages as well as recommendations of the Patterns for each level. Similarly to the previous sheet, the use of recommended or highly recommended patterns is indicated with green color whereas the use of non-recommended patterns is warned with red color.

The sheet is compiled as follows. Safety-related (UML AP) requirements and their respective safety integrity levels are collected to a list. The Packages that contain implementing design elements for the requirements are identified based on TraceRelations (of UML AP). The SafetyPatterns, instances of which can be found from the Packages, are identified based on PatternApplications. The traceability table is printed. In the table, traceability between a requirement and a Package is presented with an arrow (↘). SILs for the Packages are determined by finding the highest SILs from the requirements that are traced to the Packages. Traceability between a Package and a SafetyPattern used in the Package is, again, presented with the arrow symbol.

Traceability: Requirements --> Packages --> (Safety) Patterns

REQUIREMENT:							
P100 Protection (SIL1)	↘						
P100IR (SIL1)		↘					
PACKAGE:	Software Safety Requirements (SIL1)	ControlStructures (SIL1)	PATTERN:	SIL 1	SIL 2	SIL 3	SIL 4
		↘	Automatic software generation	R	R	R	R
	↘		Semi-formal methods	R	R	HR	HR
	↘		Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR
	↘		Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR
	↘		Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR

Figure 4. An example generated Safety Pattern Traceability sheet.

An example Safety Pattern Traceability sheet can be found in Figure 4. According to the table, it can be seen that the example model contains 2 requirements of safety level SIL1: P100 protection and P100IR. The former one (a general safety function requirement) is traced to "Software Safety Requirements" Package and the latter one to "ControlStructures" Package. SILs required from the Packages (their contents) come from the requirements, both

 ISBN: 978-1-61208-367-4

being SIL1. Moreover, the sheet presents that the use of "Automatic software generation" has been marked in ControlStructures Package and Semi-formal methods, backward traceability, forward traceability as well as computer aided specification tool in the Software Safety Requirements Package. According to the table (color coding), the techniques are recommended for the safety integrity level (SIL1) required from the Packages.

VI. DISCUSSION

This paper has presented an approach to extend the information content of design pattern concepts of UML AP with safety aspects. The new concepts enable specifying the applicability of SafetyPatterns, i.e., design patterns of safety systems, to applications of different safety integrity levels. In addition, SafetyPatterns can be collected to SafetyCatalogues with which it is possible to model both recommendations of safety standards and custom catalogues of SafetyPatterns.

To illustrate the use of the concepts, the paper has presented 3 example documentation sheets. The sheets were generated automatically based on a library model containing two SafetyCatalogues and a model utilizing the patterns of the catalogues. The first of the sheets presented one of the catalogues. The other two sheets presented compliance of a model (of a developed systems) to the other catalogue. The new information content of SafetyPatterns was in the sheets used for automating identification of safety-related patterns and consistency checks with respect to safety levels. The sheets, thus, documented rather the developed systems than SafetyPatterns themselves. In the developed metamodel, SafetyPatterns share most of their information content with the design pattern modeling concepts that are used in [6].

The authors believe that the possibility to export documentation from models is a future research topic within MDD research. Moreover, it could improve the applicability of the MDD techniques to safety system development. This is because safety applications cannot be used in practice without appropriate documentation. Without automated support for producing documentation, it would have to be produced manually. On the other hand, by automating even part of the work, it would be possible to obtain additional, MDD specific benefits in the application area.

When developing safety applications with MDD techniques, the development process should be supported. A tool should assist developers by pointing out the issues that need to be addressed, by presenting the alternatives (when appropriate) and by documenting the decisions for later use. For example, the supported process could start from modeled requirements that determine the required integrity levels. A developer could select a SafetyCatalogue to be used to guide, e.g., architecture design. Based on the selection and required integrity levels, the tool could suggest patterns to be used. In practice, this scenario could be supported with only a small modification to the Safety Catalogue sheet, by hiding inappropriate patterns based on required integrity levels.

Work that aims for guiding development work in MDD has been previously carried out by the authors also based on use of an Architecture Knowledge Management (AKM) platform [28]. Use of an external tool, however, may lead to redundant information. On the other hand, it is believed that documentation and guidance support should be available for both architectural and detailed design levels. Thus, it is feasible to integrate the required support in one tool, which is used throughout the MDD process.

A challenge in developing guidance for MDD is that development processes, techniques and solutions vary between companies and between controlled processes. The approach presented in this paper could improve the situation. Documentation sheets can be developed to support various purposes and processes, not only the ones presented in this article. In addition, by using, e.g., the SafetyCatalogue concept, the generated sheets and their contents are also dependent of the catalogues to the contents of which the models are compared. Thus, to support another kind of a development process or other techniques, one could specify other catalogues to which the models would be compared.

The authors regard safety aspects important for also basic control systems that are not critical. Safety is an issue that should be taken into account in development of any control system. Safety standards state their recommendations on techniques, measures and solutions based on evidence on their usefulness. It is likely that adopting selected techniques and measures from safety system development, e.g., traceability could also improve the quality of basic control systems. This could in turn improve the productivity of the controlled processes at least in application domains in which the development processes are not strictly governed.

On the other hand, considering selected aspects of safety standards in development of basic control systems could shorten the gap between the systems. Safety systems and basic control systems are currently not only separated from each other but also developed with different development processes and tools and often by different teams. It is possible that professionals are not even aware of the practices in the other teams. Because the development of safety systems is regulated by authorities, the only possibility to shorten the gap would be to adopt suitable practices of safety system development to basic control system development.

VII. CONCLUSIONS

Design patterns document solutions and capture expert knowledge to recurring challenges in design and development work. On one hand, design patterns support the re-use of design by preserving named, proven solutions to recurring challenges. However, they can also increase the documentation value of models that usually tend to present design solutions rather than rationale behind the solutions. With use of patterns, designs become easier to understand and the roles of design elements clear for possible third parties that use the documentation. Especially the use of patterns could benefit MDD in which the idea is to use models for both development and documentation purposes.

In this paper, a set of pattern modeling concepts was presented that enable increasing the information content of design patterns with applicability to safety integrity levels. The new concepts enable constructing catalogues of safety-related patterns with which it is possible to model

 ISBN: 978-1-61208-367-4

recommendations of safety standards. Automated functions for generating documentation sheets enable the use of the concepts for producing documentation. In addition to presenting which patterns are used in a model, the sheets present whether the models comply with the catalogues, e.g., recommendations of safety standards. The sheets can be used also during development as guidance to present the standard-compliant selections that still have to be addressed.

Ability to use models as documentation or to produce documentation from models to a suitable form is a possible key for industrial acceptance of MDD techniques in safety system development. Without automated support, the documentation would have to be produced manually. This could significantly reduce the potential to benefit from MDD. However, with documentation support, MDD would provide another means to benefit from the use of models.

When developing safety applications with MDD techniques, the development process should be supported and guided in a flexible manner. Instead of only predefined forms and checks, the presented documentation tables are compiled with use of modelled SafetyCatalogues to which models are compared. As such, the suggestions that the tool can be considered to provide are also dependent on the modelled catalogues. Tailoring the approach for different application domains or development practices could thus be possible to achieve with changes to the catalogues. While acknowledging that the development concepts still require further development, the authors regard this kind of flexibility as an important feature in MDD tool support.

REFERENCES

[1] C. Alexander, S. Ishikawa, and M. Silverstein, "A pattern language: towns, buildings, construction", Oxford University Press, 1977.

[2] C. Alexander, "The timeless way of building", Oxford University Press, 1979.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Elements of reusable object-oriented software", Addison-Wesley, 1994.

[4] OMG, "Model Driven Architecture (MDA) Guide", Object Management Group, 2003.

[5] IEC, "61508 functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements", International Electrotechnical Commission, 2010.

[6] T. Vepsäläinen and S. Kuikka, "Design pattern support for model-driven development", in 9th International Conference on Software Engineering and Applications, 2014. (in press)

[7] OMG, "Unified Modeling Language Specification 2.4.1: SuperStructure", Object Management Group, 2011.

[8] G. Sunyé, A. Le Guennec, and J. Jézéquel, "Design patterns application in UML", in Proc. of 14th European Conference on Object-Oriented Programming, 2000, pp. 44-62.

[9] No Magic, Inc. MagicDraw, 2014. Available: http://www.nomagic.com/products/magicdraw.html [retrieved: 07, 2014]

[10] R. B. France, D. Kim, S. Ghosh, and E. Song, "A UML-based pattern specification technique", IEEE Transactions On Software Engineering, vol. 30, pp. 193-206, 2004.

[11] J. Dong, Y. Sheng, and K. Zhang, "A model transformation approach for design pattern evolutions", in Proc. of 13th Annual IEEE International Symposium and Workshop On Engineering of Computer Based Systems, March 2006, pp. 80-92.

[12] P. Kajsa and L. Majtás, "Design patterns instantiation based on semantics and model transformations", in SOFSEM 2010: Theory and Practice of Computer Science, Springer, 2010, pp. 540-551.

[13] W. Xue-Bin, W. Quan-Yuan, W. Huai-Min, and S. Dian-Xi, "Research and implementation of design pattern-oriented model transformation", in 2nd International Multi-Conference on Computing in the Global Information Technology, 2007.

[14] J. Dong and S. Yang, "QVT based model transformation for design pattern evolutions", in Proc. of 10th IASTED International Conference on Internet and Multimedia Systems and Applications, 2006, pp 16-22.

[15] S. Wenzel and U. Kelter, "Model-driven design pattern detection using difference calculation", in Proc. of 1st International Workshop on Pattern Detection for Reverse Engineering, October 2006.

[16] M. L. Bernardi, M. Cimitile, and G. A. Di Lucca, "A model-driven graph-matching approach for design pattern detection", in Proc. of 20th IEEE Working Conference on Reverse Engineering, 2013, pp. 172-181.

[17] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring", IEEE Transactions On Software Engineering, vol. 32, pp. 896-909, 2006.

[18] A. Pande, M. Gupta, and A. K. Tripathi, "A new approach for detecting design patterns by graph decomposition and graph isomorphism", in Proc. of 3rd International Conference on Contemporary Computing, Springer, 2010, pp. 108-119.

[19] B. P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems. Addison-Wesley, 1998.

[20] R. Hanmer, Patterns for Fault Tolerant Software. John Wiley & Sons, 2013.

[21] T. Saridakis, "Design patterns for checkpoint-based rollback recovery," in Proc. of 10th Conference on Pattern Languages of Programs (PLoP), Spetember 2003.

[22] T. P. Kelly and J. A. McDermid, "Safety case construction and reuse using patterns. in Proc. of 16th International Conference on Computer Safety and Reliability, Springer, 1997, pp. 55-69.

[23] W. Herzner et al., "Model-based development of distributed embedded real-time systems with the decos tool-chain," in Proc. of 2007 SAE AeroTech Congress & Exhibition, 2007.

[24] G. Zoughbi, L. Briand, and Y. Labiche, "Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile", Software & Systems Modeling, vol. 10, pp. 337-367, 2011.

[25] F. Buschmann, R. Meunier, H. Rohnert, P Sommerlad, and M. Stal, "Pattern Oriented Software Architecture: A System of Patterns". John Wiley & Sons, 1996.

[26] T. Vepsäläinen, D. Hästbacka, and S. Kuikka, "Tool support for the UML automation profile - for domain-specific software development in manufacturing", in Proc. of 3rd International Conference on Software Engineering Advances, October 2008, pp. 43-50.

[27] T. Vepsäläinen and S. Kuikka, "Towards model-based development of safety-related control applications", in the 16th IEEE International Conceference on Emerging Technologies & Factory Automation, September 2011.

[28] T. Vepsäläinen, S. Kuikka, and V. Eloranta, "Software architecture knowledge management for safety systems", in the 17th IEEE International Conceference on Emerging Technologies & Factory Automation, September 2012.

 ISBN: 978-1-61208-367-4

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3528-4
ISSN 1459-2045