Klaus Pohl
Harald Hönninger
Reinhold Achatz
Manfred Broy (Eds.)

Model-Based Engineering of Embedded Systems

The SPES 2020 Methodology

Springer

Model-Based Engineering of Embedded Systems

Klaus Pohl • Harald Hönninger • Reinhold Achatz
Manfred Broy
Editors

Model-Based Engineering of Embedded Systems

The SPES 2020 Methodology

Springer

Editors
Klaus Pohl
paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
Essen
Germany

Harald Hönninger
Robert Bosch GmbH
Schwieberdingen
Germany

Reinhold Achatz
ThyssenKrupp AG
Corporate Center Technology, Innovation & Quality
Essen
Germany

Manfred Broy
Fakultät für Informatik, Lehrstuhl für Software & Systems Engineering
TU München
Garching
Germany

ISBN 978-3-642-34613-2 ISBN 978-3-642-34614-9 (eBook)
DOI 10.1007/978-3-642-34614-9
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012952094

ACM Computing Classification (1998): D.2, C.3, J.2, J.3, J.7

Preface

Embedded systems — opportunities and challenges

Embedded systems have long become an essential part of our everyday life. They control essential features in our cars, such as airbags, braking systems, or power locks, and are used to manage our steadily increasing communication needs by means of Internet routers or cell phones. Embedded systems are essential in application areas where human control is impossible or infeasible, such as adjusting the control surfaces of aircraft or controlling a chemical reaction inside a power plant. The embedded systems industry has therefore become a multibillion euro industry.

The development of modern embedded systems is becoming increasingly difficult and challenging. Issues that greatly impact their development include the increase in the overall system complexity, their tighter and cross-functional integration, the increasing requirements concerning safety and real-time behavior, the need to reduce development and operation costs, as well as the need for shorter time-to-market.

Need for an integrated development approach

Many research contributions and development methods aim to address these challenges, and theories for the seamless development of embedded systems have been proposed. However, these solutions address only a small subset of the above-mentioned problems, are only applicable in very specific settings, and lack an appropriate cross-domain validation in representative industrial settings.

The mission of the Software Platform Embedded Systems 2020 (SPES 2020) project was thus to focus on the professionalization of a cross-domain, model-based development method for embedded systems. SPES 2020 is an innovation alliance project sponsored by the German Federal Ministry of Education and Research. In SPES 2020, 21 partners from academia and industry have joined forces in order to develop a modeling framework that is based on the latest state-of-the-art in embedded systems engineering, addresses specific development challenges, and is validated in different domains to ensure its applicability in industrial embedded systems development.

Aim of this book

Industry challenges, principles, and application

The purpose of this book is to present an overview of the SPES modeling framework and to demonstrate its applicability to embedded system development in various representative industry domains. The book provides a comprehensive explanation of the basic solution concepts of the SPES modeling framework and illustrates the application of these concepts in five application domains (automation, automotive, avionics, energy, and healthcare). The book summarizes the lessons learned, outlines evaluation results, and describes how the SPES 2020 modeling framework can be tailored to meet domain-specific and project-specific needs.

Target audience

Researchers, practitioners, and consultants

This book is aimed at professionals and practitioners who deal with the development of embedded systems on a daily basis. This includes developers, requirements engineers, software or hardware architects, business analysts, mechatronics experts, safety engineers, testers, and certifiers. It serves as a compendium for researchers in the field of software engineering and embedded systems, regardless of whether you are working for a research division of a company or are employed with a university or academic research institute. For teachers and consultants, it provides a sound foundation in the basic relationships and solution concepts for engineering embedded systems and illustrates how these principles and concepts can be applied in practice.

Content of this book

This book is structured into four parts and 18 chapters:

Status quo and industry requirements

- *Part I – Starting Point:* This part discusses the status quo of embedded system development and model-based engineering and summarizes the key requirements faced when developing embedded systems in different application domains. Chapter 1 gives detailed insight into the role of embedded systems and outlines the scope of the SPES 2020 project. Chapter 2 discusses and summarizes the requirements for the development of future embedded system development in the automation, automotive, avionics, energy, and healthcare application domains.

The SPES modeling framework and its viewpoints

- *Part II – The SPES modeling framework:* This part describes the backbone of SPES 2020 — the SPES modeling framework. Chapter 3 derives the core principles of the SPES 2020 modeling framework and illustrates how these principles help in fulfilling the

requirements outlined in Chapter 2. It outlines the overall SPES modeling framework and describes the basic solution concepts of the SPES 2020 abstraction layers and viewpoints. Subsequently, Chapters 4 through 7 describe the requirements, functional, logical, and technical viewpoints of the SPES modeling framework. Each chapter defines the specific engineering artifacts subsumed by the viewpoint, outlines the basic relationships of those artifacts with the other viewpoints, and describes the engineering process across the basic abstraction layers. Chapters 8 and 9 describe how the SPES modeling framework addresses the crosscutting aspects "safety" and "real-time."

Evaluation strategy and application details

- *Part III – Application and Evaluation of the SPES Modeling Framework:* This part describes the validation steps taken to ensure that the requirements outlined in Chapter 2 are met by the solution concepts proposed in Part II. Chapter 10 outlines the overall evaluation strategy used to assess the applicability of the SPES modeling framework. Chapters 11 through 15 describe the use of the SPES modeling framework in the five application domains automation, automotive, avionics, energy, and healthcare. Each of these chapters briefly characterizes the specifics of the application domain and shows how the SPES modeling framework can be tailored with regard to these characteristics. In addition, each chapter outlines the evaluation activities conducted in the application domain by various partners and summarizes the key evaluation results. Chapter 16 summarizes the overall evaluation results and discusses them in the context of the requirements outlined in Chapter 2 and the SPES principles described in Chapter 3.

Achieved results and future work

- *Part IV – Impact of the SPES Modeling Framework:* This part assesses the impact of the SPES modeling framework. Chapter 17 summarizes the key lessons learned in SPES 2020. Chapter 18 concludes this book by providing insights into open challenges for the engineering of software-intensive embedded systems.

For further reading, a list of relevant, advanced literature providing deeper insights is given at the end of each chapter.

Acknowledgements

There are many people who have contributed significantly to this book.

Firstly, we would like to thank the members of the Steering Committee of the SPES 2020 project for their guidance and support

throughout the entire project and for encouraging us to document the project results in this book.

Secondly, we would like to thank Ottmar Bender, Dr. Wolfgang Böhm, Dr. Martin Feilkas, Peter Heidl, Dr. Stefan Henkler, Dr. Ulrich Löwen, Andreas Vogelsang, and Dr. Thorsten Weyer for their relentless efforts in integrating the different project activities that took place concurrently, for many fruitful discussions and suggestions, and for their critical reviews of project milestones. Much of the content of this book is a result of their devotion and attention to detail.

Thirdly, we would like to thank each and every author of the individual chapters for their patience in the book writing process, their willingness to revise their chapters time after time, and their cooperation and help in making this book a consistent and integrated product.

Last but not least, we would like to express our deepest thanks to Bastian Tenbergen and Dr. Thorsten Weyer for their tremendous effort and support in the overall editing process.

The results presented in this book have been made possible through the funding received from the Federal Ministry of Education and Research (BMBF) of the Federal Republic of Germany under grant number 01IS08045. In particular, we would like to thank Prof. Dr. Wolf-Dieter Lukas, Dr. Erasmus Landvogt, and Ingo Ruhmann (all with the BMBF). In addition, we would like to thank Dr. Michael Weber of the German Aerospace Center (DLR) for supporting this project.

Furthermore, we would like to thank Tracey Duffy for her valuable language editing assistance and Ralf Gerstner from Springer for his continuous support in publishing this book.

Klaus Pohl
Harald Hönninger
Reinhold Achatz
Manfred Broy

Summer 2012

Table of Contents

Part I

Starting Situation

Klaus Beetz
Dr. Wolfgang Böhm

1

Challenges in Engineering for Software-Intensive Embedded Systems

As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now that we have gigantic computers, programming has become a gigantic problem.

Edsger W. Dijkstra, ACM Turing Award Lecture, 1972 [Dijkstra 1972]

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_1,

1.1 Core Value of Embedded Systems Development

Embedded systems in everyday life

It is hard to find another market in information technology that shows similar steady growth to the market for embedded systems. As microcontrollers, they have taken over a variety of functions in multitudinous technical systems, for example, in manufacturing plants, medical equipment, power supply systems, aircraft, and cars, but also in home appliances such as washing machines and refrigerators. Microcontrollers monitor and control the systems they are embedded in. In doing so, they interact directly with their environment via communication devices or indirectly via sensors that capture data such as temperature or movement, as well as with actors that transform those data into action. Embedded systems play a key role in many high-tech sectors: they are essential in modern transportation systems — from cars, to railway vehicles, to aircraft. They automate manufacturing plants for all businesses, and are an integral part of powerful medical equipment.

Market for embedded systems

Embedded systems are microcontrollers that are connected to complete systems via sensors, actors, operator controls, and communication devices. They interact in various ways with their environment, offering a variety of functions through comprehensive software. Ninety-eight percent of the microcontrollers produced worldwide are employed in embedded systems. The programs that are executed by these embedded systems, known as "embedded software," represent an essential part of these systems and define their functionality decisively.

Embedded systems as innovation drivers

The "Nationale Roadmap Embedded Systems," issued by the German "Zentralverband Elektrotechnik- und Elektroindustrie e.V." (ZVEI) in 2008, forecasts an average annual growth of 9-10% for the embedded systems market in the coming years [ZVEI 2008]. The estimated volume of the worldwide market for embedded systems amounts to around 60 billion euros. For the most important part of embedded systems, the software, the study shows a significantly higher growth than for the hardware part.

Key competence

In businesses highly affected by mechanical and electrical engineering, software has become the most important innovation driver: a current mid-range car employs more than 70 embedded systems. The functions of the anti-lock braking system or the controls of the engine's ignition point are determined by the software of the respective embedded systems.

The capability of developing high-quality, target-driven embedded systems is a competency Germany needs in order to maintain and develop a leading position in economically important industry sectors such as vehicle manufacturing/automotive engineering, aviation, plant engineering, and automation technology, as well as medical engineering. Germany's export strength is mainly on technologies in which embedded systems represent a core value. The control of increasingly powerful and extensively networked, and as a consequence more complex embedded systems is a huge scientific and technical challenge. Mastering these crosscutting competencies offers opportunities in many application areas. Shortcomings, however, inevitably lead to risks and finally to loss of markets. It is essential, therefore, that Germany positions itself strategically in this field.

Germany is well known for the high quality of its products, mainly in areas such as automotive engineering, aviation, automation technology, and medical engineering, where embedded systems are deployed. To maintain this image with respect to embedded software as well, we have to make the same high demands on embedded systems as on other technical systems with the seal of quality "Made in Germany."

1.2 The Future of Embedded Systems

The future of embedded systems is determined by several trends that we can already recognize today:

- Increasing computing power of the systems together with the ability to store an almost unlimited amount of data at extremely low costs. This development, which we already know from computer systems, is now finding its way into embedded systems.

Increasing computing powers

- Future embedded systems will be more and more networked. Networking mainly via the Internet, arguably the most important development since the invention of the letterpress, but also using a wide variety of different networking technologies, will multiply the intrinsic intelligence of embedded systems

Networked systems

- The increasing structural and functional integration with mechanical and electrical system parts will finally produce "cybertronic systems," consisting of mechanical, hydraulic, or pneumatic parts, as well as sensors, actors, and information processing units linked to each other via flows of energy, material, or information. The line between mechanical components and software system will become more and more blurred. The software part of "embedded systems" is

Structural and functional integration

shifting from an enabling technology towards a core technology: it forms the products to which it belongs. The software part transforms, enriches, and becomes the dominant part of the new generation of products [Beetz 2010].

High complexity

Future embedded systems will be characterized by high complexity. Compared to pure mechanical or electrical systems, future embedded systems will have a much higher number of coupled elements (networking). In addition, these elements will be implemented by different technical disciplines involving different types of coupling (mechanical, electrical, IT). More and more software will be crucial for the functionality and the coupling of the individual mechanical and electrical components. This coupling will finally lead to the tight integration of today's isolated engineering disciplines of mechanical engineering, electrical engineering, and computer science.

The development leads from closed, well-arranged embedded systems that can be found in coffee machines, ATMs, or heart pacemakers, to systems that become more and more intelligent, to systems that can build their own intentionality, enabling them to make goal-oriented decisions on their own and to act accordingly. Ancestors of those future embedded systems are already among us today as helpful servants that can cut trees for us, prepare sandwiches, assemble cars, fly airplanes, or explore distant planets.

Integration with classical information and communication technology

As a consequence of the integration of embedded systems with classical information and communication technology and the Internet, systems will arise that span the globe, working together seamlessly and forming separate "digital spheres" such as global intelligent shells consisting of android buildings, factories, hospitals, transportation systems, up to highly automated agriculture and an inexhaustible knowledge base. The dream of the famous encylopedists of the age of enlightenment *Denis Diderot* and *Jean-Baptiste le Rond d'Alembert,* to make all the knowledge of the world accessible and useable to everybody, seems to be coming true, because the Internet age has just begun.

Our knowledge is being transferred to the Internet at full speed. Eric Schmidt, former Google CEO and now chairman of the board, illustrated the tremendous storage capacity of the Internet as follows: "In 2029 you will be able to buy eleven Peta-Bytes (quite a big number) of digital storage on a single hard drive for less than 100 Dollar. According to my calculation this device will be able to store every single day, 24 hours in DVD-Video quality for six hundred years" [Schirrmacher 2011].

Digital spheres

No knowledge, no experience that cannot—and will not—be recorded. However the true value of information is not based on the

information itself, but on the networking and the algorithmic evaluation of the information stored. Therefore, the "digital spheres" not only contain information and knowledge, but moreover awareness of how to use the information and the knowledge, which in turn will change the information and the knowledge permanently.

Thus, a separate digital parallel world is arising, to which the term "artificial intelligence" is probably more applicable than to the traditional field of science that is aimed at emulating the human brain. Humans will be part of this intelligent parallel world and they will have to shape and control it. We will not discuss the social and political consequences here, but they will drastically change our lives, especially our professional life and the way we live together.

Examples of these "digital spheres" are the global communication network, the huge worldwide information marketplace of the financial world, and the Global Positioning System (GPS) that has already become an integral part of transportation systems. In the near future, the power grid will also build such a "digital sphere." This "smart grid" will be based on the most advanced embedded systems available. By virtue of distributed intelligence alone, the "smart grid" will be able to guarantee predictable stability and functionality of the electrical energy network. This development will not pass by many engineering disciplines, as something like "Google Engineering" will arise, where architecture and design decisions will be taken from the endless resources of the "digital sphere."

Cyber-physical systems

In the US, the term "cyber-physical system" is used for those systems, forming the basis of the "digital spheres." In a cyber-physical system, electronic systems are interlaced intelligently with network components and physical systems in a way that integrates the physical systems to give them new capabilities. The acatech (German National Academy of Science and Engineering) project "Agenda CPS" develops a general view on the political, economic, technical, and research challenges of cyber-physical systems. In this project, cyber-physical systems are understood in a broader sense, including issues around connecting embedded systems to global networks such as the Internet.

Embedded systems are the important building blocks of cyber-physical systems and are used by the various digital spheres. Here they implement in particular the interfaces of the "digital spheres" to the user and to the technical and physical components, thereby forming the link between the virtual "digital spheres," the humans, and the real world. Mastering the complexity this introduces constitutes a central challenge. In his keynote speech at the ITEA (Information Technology for European Advancement) symposium 2010, the Chairman of ITEA,

Rudolf Haggenmüller, coined the phrase "embedded hardware." This comprises hardware in the primary sense, i.e., not microcontrollers or control units, but also in the sense of the scenarios described above. Embedded hardware could be a car, a building, a milking machine, or even a complete power plant — therefore, hardware that can be understood as embedded in a complex network, in a digital sphere.

The foundation for mastering the challenges of the development of embedded hardware is established by the SPES 2020 project [BMBF 2009].

1.3 Vision of SPES 2020

National innovation alliance

As outlined above, developing software for the increasingly more powerful, more internetworked, and as a consequence more complex embedded systems is a huge challenge. Therefore, 21 partners from industry and science have formed the national innovation alliance "Software Platform Embedded Systems 2020," targeted at making the production of embedded software across industry domains professional by means of an integrated and powerful methodology [Beetz 2010, Broy 2010].

The vision of SPES 2020

Mastering the related challenges represents an important advantage for German products in European and international markets, and is therefore essential for job creation and welfare. This highlights the enormous benefit of such a concentration of research and development work, especially because of the numerous application areas in key German industries. It is the vision of SPES 2020 that in the near future, it will be possible to develop embedded systems, containing a high amount of embedded software, using a set of integrated modeling techniques whose interdependencies and cooperation are completely understood. SPES 2020 envisions that:

- The functional and nonfunctional requirements of such systems can be completely modeled at system level using appropriate abstraction.
- Analysis, verification, and validation steps can be performed based on those models.
- Decomposition for the interface behavior of the systems in the sense of architecture and a more step-by-step realization of a technical architecture can be derived from these functional models.

Uniform modeling techniques

We will still split the systems into mechanical, electrical, and technical parts. In doing so, we will use uniform modeling techniques for all three disciplines, or at least clearly defined, standardized interfaces that cover

all aspects of these three different system parts and describe their interactions in a modular fashion and in the sense of a compositional modeling technique.

High degree of automation

The main goal is to formalize the models to the extent that a high degree of automation is possible, including consistency checks and generation of validation methods to make the systems quantifiable and their properties explicitly represented and documented. These methods can be either tests or logical analysis. The planned high degree of automation will make it possible to generate software out of its abstract and generally modeled properties for different and sufficiently specified platforms. This of course requires the modeling and specification of the platform properties in sufficient detail.

Concept of reuse

The longer-term vision is the availability of a comprehensive concept for reuse for the respective application areas. Based on given, reusable platforms and system building blocks, it should be possible to execute major parts of the development work by utilizing predefined and domain-specific specifications, building blocks, and reference architectures.

1.4 Mission of SPES 2020

Cross-domain development

The national innovation alliance “Software Platform Embedded Systems 2020” (SPES 2020) thematically follows the aim of professionalization of a cross-domain development process, mainly the classical targets of software engineering: productivity and quality.

The focus of the research and development work is in embedded software, which is widely ramified into the areas of mechanics and electronics in order to leverage the comprehensive optimization potential arising from those disciplines. The goal of SPES 2020 was to create a unique innovation alliance in Germany that works across application domains to develop future networking, hardware, and software architectures, as well as new methods for software and system engineering.

Integrated model-driven and tool-supported approach

A model-driven and tool-supported approach that is based on a strong mathematical foundation allows for the efficient development of embedded systems, starting with initial customer requirements, through specification of architectures, through implementation, to system verification and certification.

This objective required a lot of work to be done in applied research as well as in fundamental research to complete relevant results. SPES 2020 was able to provide an integrated approach for model-based development of discrete systems with a strong emphasis on interfaces, distribution,

and interaction, as well on a consistently architecture-centric approach that includes adequate system structuring and a model of the individual architecture elements.

Joining the different modeling approaches

The basic terms, concepts, and theory have been available in results of fundamental research that has been more or less complete in that area. SPES managed to join the different modeling approaches supported by the academic partners. However, the main focus of SPES has been the transformation of those modeling approaches into usable terms and characterization techniques needed for an engineering approach, and the consolidation of methods for the creation of the various model views for an integrated use for target-oriented system and software development. The different, more pragmatic approaches of the application domains have been recognized.

Domain-specific approaches

The domain-specific approaches have been simplified and consolidated as far as possible, and a solid understanding has been developed with regard to which application domains really require specific modeling techniques and how advanced modeling approaches in the different domains can be brought to the general modeling approach.

Therefore, the SPES central project has taken up the available approaches from the application domains and merged them with results from scientific research. The resulting modeling methods have been reflected back to the application domains using case studies. Additional requirements from the application domains have been collected and have been considered in the definition of the modeling methodology.

Properties of the environment

A special challenge that was not the main focus of SPES was the modeling of system properties that are not classical software properties but properties of the environment (e.g., electronic or mechanical) in which the systems are embedded. In the SPES 2020 context, we restricted the work to the question of which properties of the environment are necessary, what the interfaces to the domain models look like, and how those models should be brought in to guarantee consistency between the physical models and the models of the embedded software.

A special focus in the project was on the integration of model-based development in order to leverage the synergy that comes from integrating the models across a larger number of development steps instead of looking at the models of each step in isolation.

Architecture as central artifact

Architecture is the central artifact in model-based development. In SPES, the term “architecture” is understood very broadly as any kind of structuring system from a functional view, logical component view, or technical view, with a focus on software and hardware architecture, as well as on deployment and scheduling.

Last but not least, an important goal of SPES 2020 was to provide an adequate concept for tool support based on the integrated approach for model-based development that was developed. Prototypes of this concept have been developed and tested.

With these achievements, the innovation alliance SPES 2020 has significantly strengthened the strategic and completive position of Germany as a leading country for the development of embedded systems. But to maintain this position as a leading engineering country and create new jobs and welfare for society, further effort is necessary from all stakeholders, from academia, from industry and from public authorities. Examples of future topics to be addressed by the innovation alliance are the managing of variants and multidomain engineering.

1.5 Research Approach

As discussed above, a central idea of model-based development of embedded systems is the integration of different approaches from the application domains and consolidation of these approaches to form an integrated model-based approach that has the potential to be a comprehensive tool support that can be deployed in various application areas.

By nature, this is a difficult and comprehensive task to which scientific foundations, historically grown views, and different requirements from the various application areas contribute. In addition, the participating scientific groups follow their own approaches, starting from different theories. Up until now, integration at theory level has not been completed, and a series of foundation work has yet to be done.

Four main focus points

Against this background it was important to gain a clear vision of how the different views and approaches in SPES could be integrated. This gave four main focus points for the development of an integrated approach for model-based development.

1.5.1 Generic Overall Approach — Metamodel

Fundamental ideas and concepts

The generic overall approach identifies the fundamental ideas and concepts of how to proceed. These include the use of different views and viewpoints in the sense of the specification of a series of abstraction layers for modeling the architecture. These architectural layers are in documented verbally and in approximate terms, represented graphically, and finally mapped onto a metamodel. In the generic layer, the basic philosophy is described without a detailed theoretical elaboration, and

more pragmatically, without a concrete practical implementation with respect to tool support or concrete modeling techniques. It is important that not only the philosophy of the architectural approach and its modeling framework are described, but also that the whole engineering process is captured.

Three approaches to make the generic approach more concrete are outlined below.

1.5.2 Fundamental Scientific Approach

Comprehensive theory

A rigorous, theory-based scientific approach to substantiating the generic approach is to work out a comprehensive theory of model-based development, including all theoretical investigations and elaborations, so that for all concepts and ideas described in the generic approach, there is a comprehensive scientific elaboration and theoretical foundation. This ensures that all concepts of the generic approach are completely analyzed, elaborated, and justified from a theoretical perspective.

The theoretical framework will show that all the concepts are consistent, complete, and fit together seamlessly. This lays the foundation for a methodology that is theoretical in spirit but not immediately deployable from a practical perspective, and that completely answers all questions concerning modeling and specification, not necessarily claiming that these concepts will immediately scale or can be put into practice. Consequently, this theoretical approach offers strong momentum for the concrete approach, as it forms the necessary foundation for clear terminology and the proven concepts.

1.5.3 Pragmatic Implementation

Pragmatic implementation

The pragmatic implementation of the generic approach starts with existing approaches (for example, UML) existing tools (such as MATLAB or the tools from Esterel Technologies) and tries to integrate the existing, often very fractal approaches in terms of the generic approach step-by-step. In doing so, the different views defined in the generic approach are described as completely as possible by means of specification languages and tools available in practice. A tight integration is sacrificed in favor of the use of existing tools and concepts. Where necessary, certain breaches, inconsistencies, and, to some extent, missing precision are accepted. The benefit of this approach is its possibility for a fast transfer into practice, high acceptance by practitioners, and a good adoption to the existing processes in practice. The drawback is limited automation support and limited comprehensiveness.

Of course, it is possible to utilize the scientific foundation, developed by the more theoretical SPES approaches, insofar as these can already be implemented by the existing tools. There are several advantages of this approach:

Firstly, we see faster deployment of the theory, better acceptance by practitioners, and a big impact on procedures available in practice. Secondly, SPES can benefit by discovering holes in the metamodel or different philosophies in various application areas that cannot be united easily.

The drawback of the pragmatic approach is its limited support of automation as well as a limited integration.

1.5.4 Concrete Implementation

Concrete implementation

The generic approach can be mapped to a concrete implementation without adopting the partially integrated pragmatic approaches. Starting with the generic approach, a clean theory is set up as part of the rigorous foundation and clear concepts are developed. The goal is to develop an approach that is theoretically clean on one hand, and closely aligned with the needs of practice on the other. Gaps and incompleteness are recognized and accepted and will be filled on an ongoing basis. This drives a step-by-step understanding of the approach in practice. The goal is to develop a clean practice that is always aware of its gaps and incompleteness and that can be filled slowly. In contrast to the pragmatic approach described above, a rigorous approach that only accepts and deploys clean accented methods and techniques is followed. The concrete implementation serves as an intermediate step towards a well-founded scientific implementation that puts the model framework forward step-by-step, taking the scientific foundations into account.

1.5.5 Synthesis

SPES 2020 contains elements of three fundamentally different modeling approaches and relates them such that they do not compete and cripple one other, but instead have been fused in order to complementing one other. It should be noted that integration between the approaches is not possible; integration within the different approaches would be better. However, the approaches can benefit from each other by trying to translate between the different layers to highlight and use the relationship concepts.

1.6 Topics Not Addressed in SPES 2020

Advanced topics

A series of questions regarding more advanced topics related to embedded software systems have not been addressed in SPES, such as autonomy, adaptivity, and self-organization. Furthermore, questions related to future hardware architectures, such as multicore architectures, have only been explored to an extent that was necessary to verify the appropriateness of the SPES development paradigm for these new hardware architectures. Innovative architectures for embedded systems have also not been explored.

Within the research leading towards the SPES modeling framework, product line engineering was not considered explicitly. This is because it would appear that this additional dimension of variability can only be investigated once the basics of model-driven development have been comprehensively worked out.

However, the project has developed systematic methods for the architectures in the development process, such as reference architectures and architecture frameworks, as well as their role in the development process. Advanced topics, such as product line architectures and systematic reuse, are the focus of future work.

1.7 References

[Beetz 2010] K. Beetz: Was wird besser, wenn SPES erfolgreich ist? White Paper, 2010.

[BMBF 2009] Bundesministerium für Bildung und Forschung (BMBF), Referat Öffentlichkeitsarbeit: SPES 2020 Software Plattform Embedded Systems 2020, Infoblatt, 2009.

[Broy 2010] M. Broy: Mission und Vision von SPES 2020. White Paper. 2010.

[Dijkstra 1972] E. W. Dijkstra: The humble programmer. In: Communications of the ACM, Vol. 15, No. 10, 1972, pp. 859-866.

[Schirrmacher 2011] F. Schirrmacher: Wir brauchen eine europäische Suchmaschine, Frankfurter Allgemeine Zeitung, 19.07.2011, No. 165, p. 27.

[ZVEI 2008] Zentralverband Elektrotechnik- und Elektroindustrie e.V. (ZVEI): Nationale Roadmap Embedded Systems, 2008.

Ottmar Bender
Martin Hiller
Bastian Tenbergen
Dr. Thorsten Weyer

2

Requirements from the Application Domains

This section provides an overview of the current situation of embedded systems development in the individual domains that participated in the SPES 2020 project. Then, based on the current situation, the section develops concrete requirements for a continuously model-based development process for embedded systems for each domain. The section concludes with a report on an empirical analysis of these requirements.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_2, © Springer-Verlag Berlin Heidelberg 2012

2.1 Initial Situation in the Application Domains

Embedded systems are different in each application domain

Embedded systems are different in every application domain. These differences arise due to the contexts in which the embedded systems will be deployed. For example, while an engine control unit is embedded in a car, a pacemaker is embedded within a human being. However, not only the context of the embedded systems, but also the constraints under which development takes place are different in each domain, even for each development project. For example, in the avionics and healthcare domains, very strict safety standards and certifying authorities govern the process as well as the product to be developed.

On the following pages we discuss initial situations within each application domain. The intention is to shed some light on current development situations in which embedded systems are being developed. Based on these situations, concrete requirements for a continuous model-based development process can be elicited, as discussed in Section 2.2.

2.1.1 Initial Situation in the Automation Domain

In the automation domain, embedded systems can be found at different levels. There are *devices for automation,* including intelligent sensors and actuators, that are typically presented and offered to the customer in a product catalog. On the one hand, there are standard devices such as programmable control devices or standardized power trains. On the other hand, there are also numerous special purpose devices for specific problems in automation: *machines (e.g*., *robots)* that are either provided to the user in a product catalog or developed individually for customers, or *partial constructions* (e.g., a product line) or entire constructions that are developed individually for a specific user.

Bespoke system development

Hence, embedded systems in the automation domain are both: systems that are primarily realized in development projects independent of any customer contract and systems that are realized primarily in projects based on a contract with the customer. In projects with customers in particular, the usage view of how an entire embedded system is developed by means of embedded software-intensive systems and which concepts the automation devices provide to support the integration into an entire system is of vital importance.

For a *provider of automation devices*, this results in the following challenges with regard to business and technology:

- Managing the numerous variants that result from different "performance parameters," bus systems, requirements for safety, reliability, etc.

- Ensuring high quality and robustness
- Ensuring that the automation devices are capable of being integrated with each other as well as with other devices

For a *system integrator,* this results in the following challenges with regard to business and technology:

- The entire construction and essential parts of it are developed within projects with customers, which, based on experience, entails a high corporate risk.
- The development process comprises the integration of several purchased components and services, which entails a high technical risk.
- During development, different disciplines such as process technology, mechanics, electrical engineering, and software must be integrated with respect to the procedure and the work results. Since the software is integrated in the last step, this integration has the task of ensuring the correct interplay between the disciplines.

2.1.2 Initial Situation in the Automotive Domain

Steadily growing complexity

Embedded automotive systems built for today's mobility requirements are growing in complexity. Therefore, new and refined development methods are required. Across all domains, engineering for embedded systems is characterized by a physical context with real-time requirements and the need for interdisciplinary cooperation. Additionally, the automotive domain has a high proportion of quality requirements, cost pressure, and resource constraints. These result from high volumes ranging in the millions, particularly demanding safety and reliability requirements, and extensive variability stemming from a large number of system approaches and functional configurations.

Within the SPES 2020 project, the partners in the automotive domain planned to address the challenges described above with the objective of developing new or refining existing methods that ensure that systems with this level of complexity can be developed more efficiently.

2.1.3 Initial Situation in the Avionics Domain

Embedded avionics systems built for today's aircraft are growing in size and complexity, therefore new and refined development methods are required. To improve this situation, the SPES 2020 project was established. SPES 2020 aimed at developing new or refining existing

methods to ensure that systems growing in size and complexity can be developed more efficiently.

Strict certification guidelines and safety requirements

Avionics systems have to be certified by the airworthiness authorities before they can be installed and operated in an aircraft. In order to achieve certification for an avionics system, the company developing and manufacturing this system has to provide the airworthiness authorities with the evidence that the system is safe.

A system is considered safe if two prerequisites are fulfilled by the system builder: firstly, a safety analysis has been conducted that shows that the probability of hazards caused by the system is sufficiently low; secondly, the manufacturer can demonstrate that the system's hardware and software parts have been developed according to pre-defined processes and have been accepted by the airworthiness authorities. This means that methods developed or refined in SPES 2020 had to support and comply with the safety analysis and development processes in the avionics domain. The methods developed in SPES are applied in the typical development process steps: *requirements analysis*, *design*, *implementation*, *integration*, *verification,* and *validation*.

Current methods and principles of incremental certification also have to be refined to reduce certification effort for future systems. On the other hand, efficient methods are necessary for recertification of systems that have been certified once and have to be modified, e.g., due to implementation of new features.

2.1.4 Initial Situation in the Energy Domain

Networks of heterogeneous systems

It is widely understood that modern power generation will consist of a mix of generation from conventional power plants, such as nuclear or coal-fired power plants, and an increasing amount of generation from renewable energies such as wind, biomass, and solar power. The latter part of the generation mix is installed in a decentralized manner, i.e., apart from some larger power generators such as wind turbines, there are plenty of small generators with approximately 5-100 kW. Examples are photovoltaic units placed on the rooftops of residential homes or combined heat power generators.

Besides the trend towards a large number of small energy resources, we can observe that more and more often, the characteristics of devices such as inverters towards the distribution grid can be controlled electronically and are programmed into the devices themselves. This raises the question of how to set up and evaluate a massively distributed energy system taking advantage of the benefit of the “magic of large numbers” in the case of small, controllable energy resources. Due to the

small size of the distributed energy resources, the characteristics that have to be considered for managing such distributed energy systems are:

- More than 100 resources have to be managed simultaneously.
- The energy resources are often operated by local operators with local objectives (dual use).
- Engineering effort per resource is not affordable.
- Available communication infrastructure including power line communication (PLC) should be reused.
- There is no guarantee that a resource is operating in a controlled fashion.

Based on the characteristics, the following technical requirements have to be fulfilled by an implementation managing the system:

- Full self-configuration of the distributed energy resources
- Tracking of availability and operation of energy resources
- Generic modeling of the different energy resources
- Compensation of stochastic loss of control of particular energy resources
- Requirements of the particular states of the distribution grid

2.1.5 Initial Situation in the Healthcare Domain

Similarly to the avionics domain, embedded systems in the healthcare domain are governed by strict safety guidelines and standards and are required to pass certification before they can be legally operated. In particular, rules imposed by regulatory authorities such as the FDA not only have to be adhered to by the product, but in some cases by the development process as well. As a consequence, safety and regulatory concerns dominate the development process and must be considered meticulously during system development.

Safety aspects within medical systems residing in the human body

Furthermore, some types of embedded systems in the healthcare domain are particularly constrained, as they reside within a human body. Pacemakers, for instance, are subject to special constraints regarding their maintainability. Once the system is installed, changes, maintenance, or alterations are undesirable, as they would require the patient to undergo further, potentially dangerous surgery. Therefore, it must be possible to maintain these systems easily and ideally externally (if at all). In summary, systems that reside within the human body have properties that must be accounted for during development to ensure safety, maintainability, and testability.

2.2 Requirements for the SPES Engineering Approach

The SPES engineering approach is intended to address the challenges that arise in the application domains. These challenges were outlined in Section 2.1. The requirements for the SPES engineering approach are presented below. These requirements are based on the initial situations presented above.

2.2.1 Requirements from the Automation Domain

Automation must cover process technology, mechanics, and electrical engineering in the individual projects to different degrees. Therefore, the requirements for the software are very different (see, for example, the diverse security standards). From the perspective of the automation domain, the following requirements and challenges were therefore addressed within the SPES project:

- *Supporting the modeling of technical systems:* Due to the increasing complexity of systems in the automation domain, it is necessary to create adequate models in order to be able to manage the development of these systems. For this purpose, the following means are commonly used: appropriate abstractions, different views on the system, consideration of different aspects. In addition to the individual models themselves, dependencies between the different models have to be documented as well as the development process, or more specifically, the underlying development methodology used to create or refine the models during development. In order to be able to exploit all the benefits of model creation, an appropriate theoretical foundation of the modeling approach is necessary.
- *Supporting system integration:* Due to the necessity of integrating the different systems and services when developing systems in the automation domain, the specific aim is to use the models to support this integration. This involves not only compatibility of the content of the models, but also the procedure with regard to the engineering workflow during development of the systems. Considering the systems thereby merely from a software-technical view is not sufficient, since the software is embedded in a physical system and has to be considered as an integral part of it, because typically the architecture of the software is determined by the architecture of the physical system.
- *Ensuring specific system properties right from the beginning:* To reduce the technical and business risks faced during development of

systems in the automation domain, the models shall be used to ensure essential system characteristics. Therefore, an appropriate model of the physical system in which the software is embedded is necessary (as well as the model of the software). In this way, the system can, for example, be put into operation virtually. This also requires an appropriate open infrastructure for executing the models.

- *Providing tool chains for engineering of embedded systems in the automation domain:* Due to the complexity of systems in the automation domain, their development must be supported by adequate engineering tools. Since a number of different tools are already used in practice, a main requirement was to extend these existing tools appropriately, to use them, and to integrate them into a continuous tool chain. The integration shall be driven by a consistent modeling theory for technical systems to ensure sustainable tool support.

2.2.2 Requirements from the Automotive Domain

The major requirements from the automotive domain for the SPES engineering approach are the following:

- *Supporting the systematic gathering and documentation of requirements:* The SPES modeling framework shall apply model-based requirements engineering to the automotive domain systems to enhance system understanding, provide guidance for system design, and deliver proof of fulfillment of required properties.
- *Supporting the transition between informal and formal requirements:* The SPES modeling framework should provide a systematic method for transforming a set of mainly informal requirements into an implementation that is based on the domain-specific AUTOSAR standard.
- *Supporting functional development:* The SPES modeling framework shall introduce model-based functional development throughout the automotive development lifecycle and shall integrate the discrete and continuous problem classes into a homogenous system design.
- *Supporting model-based safety design:* The approach should support model-based safety design in automotive development to achieve safety properties by design and reduce the safety validation effort.
- *Supporting the analysis of functional correctness:* The SPES modeling framework shall identify design flaws regarding functional correctness and timing in early development phases.
- *Supporting the AUTOSAR standard:* The SPES modeling framework shall transform the SPES metamodel into corresponding AUTOSAR

models in order to apply the SPES analysis techniques in the AUTOSAR context.

- *Empirical validation:* The SPES modeling framework shall validate and explore the limits of the developed methods in the automotive domain empirically to prove the effectiveness of the approaches.

2.2.3 Requirements from the Avionics Domain

The major requirements from the avionics domain for the SPES engineering approach are the following:

- *Supporting the systematic specification of requirements:* To specify consistent, understandable, and unambiguous system and software requirements, the application domains in SPES 2020 proposed to elaborate available formal or semiformal specification languages to ensure their efficient usability. An important aspect of usability is that requirements must be readable and comprehensible to engineers and representatives of certification authorities. To improve the understanding of the dependencies among the requirements, the need for a suitable modeling technique was identified for SPES 2020. A further objective in this area was to exploit the formalism in the requirements in order to generate test cases or test case fragments automatically and to utilize the formal character of the requirements to perform consistency and completeness checks on the requirements specification. The requirements engineering methods developed must adhere to different levels of certification strictness imposed by the different certification standards and authorities.
- *Supporting the systematic analysis and documentation of system architecture:* In order to develop large scale systems, for example, smart grids, rolling mills, and aircraft, a systematic refinement of architectural modeling techniques that takes different aspects (e.g., safety) into account is required. In addition, the definition of abstraction layers and the relation of artifacts between them is very important for coping with the complexity imposed by the scale of the systems considered. The optimization of possible design solutions regarding safety and performance and the utilization of the computing resources available has been stated as a further goal in SPES 2020. It has been recognized that the availability of a modeling language (e.g., SysML) is not sufficient to achieve the goals described above. In addition, efficient modeling techniques and methods are required.
- *Supporting continuous modeling of safety and system certification:* Safety and system certification play an important role in the

application domains of SPES 2020, especially in the automotive, avionics, and healthcare domains. Today's safety analysis is performed on separate design and safety models using different tools. To avoid error-prone and inefficient redundancy of the design and safety models, a requirement for the SPES 2020 program was to define solutions allowing a safety analysis that integrates system design and safety information in one model. A further requirement for the avionics domain was to elaborate methods for automatically generating safety cases based on the system design model.

- *Supporting the verification of engineering artifacts:* The verification activities create a high workload in the application domains. The requirement for the SPES 2020 program was to reduce this workload by defining methods for generating test cases and procedures automatically based on requirements and design information.

2.2.4 Requirements from the Energy Domain

The major requirements from the energy domainfor the SPES engineering approach are the following:

- *Supporting the consideration of large numbers of massively distributed embedded components:* Smart grids are potentially large systems with huge numbers of massively distributed embedded components (up to the order of millions distributed across hundreds of square kilometers). The high number of components in real systems can cause the overall system to show characteristics that do not emerge in smaller systems with a comparatively low number of components.
- *Dealing with complexity in system structure and component interaction:* Certain events in a power grid, for example, a decrease in generated power, can cause events within the communication network, such as messages, to switch off consumers. Power generation can be affected by weather conditions (solar power, wind power). Consumer behavior influences energy demand. Energy markets and international integration of power grids influence energy transmission and financial transactions.
- *Supporting the engineering constraint "one-shot scenario":* Because of the sheer size of smart grids, many technical decisions that are taken during the concept phase are virtually irreversible after the smart grid has been realized and installed. Thus, careful planning before starting the realization phase is very important. Mistakes can lead to huge costs in later project phases. This is why it is important

to design and test smart grids by means of simulation before installation to uncover and fix problems beforehand.

- *Supporting dynamics in system structure and system behavior:* Due to the characteristics and the implementation of specific components within a smart grid, system structure and system behavior can be dynamic, for example, with respect to availability, stability, and failure resilience. For example, components within a smart grid, such as local generators and consumers, can join or leave the grid at any time. Furthermore, due to the massive distribution of components, no single entity has total control of all components in the smart grid. Failure or faulty behavior of single components within the system is thus inevitable. The overall system must be able to cope with these challenges.

2.2.5 Requirements from the Healthcare Domain

The healthcare domain faces similar challenges to the avionics domain. Most embedded devices in medical applications are safety-critical and have to pass a long and intensive certification procedure. Again, similar to avionics, in such systems a safe state cannot be reached by simply switching off the equipment. Imagine, for example, a life-supporting device where the health of the patient depends critically on the correctness of the embedded software. Thus, all components must be redundant and highly reliable. The software development process for such a system has to follow strict rules and all artifacts must be validated.

For a model-based software development, this means:

- *Safety aspect:* Safety is of utmost importance. It must be possible to explicitly state safety requirements in the models and to assess whether they are realized in the final system. Other nonfunctional properties such as usability, adaptability, and configurability can be regarded as special instances of safety.
- *Traceability:* In order to allow for effective validation, all models must be linked to each other such that requirements can be traced from the initial specification, through the various modeling stages, to the final executable code. Ideally, the models are instances of a common metamodel and have clear, unambiguous semantics.
- *Interoperability and adaptability:* Devices and processes must be interoperable and easily adaptable. This can be achieved through standardized reference architectures. The modeling methodology must provide a way of including such reference architectures (e.g., as a model library) and instantiating the standardized component for a specific project.

- *Energy efficiency and maintainability:* Since devices such as pacemakers remain within a patient for a long time, energy efficiency and low maintenance efforts are very important for certain medical devices. The methodology must provide ways of combining resource considerations with the certification needs as required by regulatory descriptions.
- *Testability:* Testability is an important assessment criterion. Therefore, the modeling framework should support automated test case derivation from models as well as code generation. There must be a clear distinction between implementation models and test models.

Model-based design has a high potential to improve the software development process of embedded medical devices. However, software development is only one aspect in the design and production of such systems. The SPES modeling framework shows how validation and certification aspects can be incorporated into the process to allow for better products and a significantly shorter time-to-market.

2.3 General Requirements from Industry

At the beginning of the SPES project, a study was conducted with representatives from companies in all application domains of the SPES project in order to gather their major requirements from industry concerning the SPES engineering approach. The participants' self-reported areas of operation included research and development (40% of the participants) as well as process and project consulting (another 40%). Of the participants questioned, 60% reported their experience with requirements engineering to span 5 to 10 years, 20% even reported more than 15 years of experience, and 90% of the participants reported their level of experience in requirements engineering as advanced or expert.

The study employed a combination of qualitative and quantitative techniques in order to yield deep insight into the state of practice and the needs concerning model-based engineering. Data was acquired by means of a structured interview and a post-interview questionnaire.

The findings from the study are summarized below and related to the focus of the SPES 2020 project. Detailed information about the motivation for the study, the feedback gained from the participants, and the detailed analysis and conclusions can be found in [Sikora et al. 2012].

2.3.1 Empirical Finding: Need for Model-Based Engineering

Natural language requirements vs. requirements models

Natural language is the most common documentation form for engineering artifacts. However, there is strong evidence that practitioners are dissatisfied with natural language, as dealing with large bodies of natural language documents is perceived as tedious and error-prone. In contrast, using models during the engineering of embedded systems is perceived as beneficial as models help in the understanding of complex engineering problems, serve as a natural means for structuring the problem space, and make communication with other stakeholders easier. As a result, in current practice, models are used to support engineering and often supplement text-based documentation of engineering artifacts. Our study showed, for example, that executable models (such as MATLAB/Simulink models), semiformal models (such as SysML [OMG 2010a] and UML [OMG 2010b] models), as well as domain-specific models from disciplines such as mechanical engineering, electrical engineering, or control engineering are common artifacts throughout the engineering process.

Artifact-based quality assurance

One of the most important purposes of these artifacts is early validation and quality assurance. However, despite the advantages of using models, many practitioners refrain from applying them. One key reason is that there is confusion about when to apply models during engineering and when to resort to traditional natural-language-based documentation, particularly when legally binding documents are involved, safety standards must be satisfied by means of models, or models are used that are applied in different engineering activities (for example, structural models that are used during requirements engineering as well as architecture design).

In summary, an engineering approach is needed that fosters model use during different engineering activities and supports the use of model types that are already common in the engineering of embedded systems.

2.3.2 Empirical Finding: Need for Artifact-Orientation

Artifact interoperability

As mentioned in Section 2.3.1, the state-of-practice study was able to confirm that natural language is the dominant documentation form for engineering artifacts. As a result, artifacts in the engineering of embedded systems are typically natural language documents that also serve as a contractual basis and are at best supplemented with models. Furthermore, as these documents are usually holistic in nature, information contained therein for specialized engineering activities such as quality assurance, architecture design, or safety engineering is hard to

discern. In particular, many engineering artifacts in current practice do not meet the prerequisites for applying automated techniques.

Therefore, approaches are needed that allow for the co-development of artifacts that can be used and re-used for a variety of engineering activities.

2.3.3 Empirical Finding: Need for Continuous Method Support

As mentioned in Section 2.3.1, there is uncertainty about when to use models to aid the engineering of embedded systems, and this is one of the key factors inhibiting more intensive model support. Another inhibiting factor is missing method support for the application of models during different engineering activities.

Artifact integration between engineering activities

While some knowledge exists regarding different model types and their suitability for different engineering activities, an approach for the seamless integration of models during engineering and the transition between engineering activities is largely missing. For example, missing method support leads to an enormous effort for ensuring the consistency between requirements engineering artifacts and safety engineering artifacts. In particular, method support is missing for specifying artifacts across a hierarchy of abstraction layers. In addition, results of the study provide evidence for a close interrelation of requirements engineering and architecture design, but also indicate some confusion regarding the separation of the resulting artifacts from both engineering activities (see Section 2.3.1). As a consequence, participants expressed a strong need for systematic support for traceability between these two engineering activities.

2.3.4 Empirical Finding: Need for Differentiation of Abstraction Layers and Transition between Them

Since the complexity of modern embedded systems is continuously increasing, new challenges for their engineering also arise. In order to meet these challenges, the development process must be structured strictly. The participants of the study stated that performing engineering across a hierarchy of abstraction layers is one of the essential means to achieving a structured development process. In particular, a systematic approach that takes the refinement of engineering artifacts into consideration is missing from current practice.

Seamless transition and integration

In addition, practitioners expressed the need for seamless transition between abstraction layers. Although abstraction layers are seen as

beneficial, confusion exists concerning their application. In particular, there is uncertainty about which engineering artifacts to define at what level of abstraction, what level of detail should be included in an artifact, how abstraction layers can be tailored to specific project needs, and how consistency between artifacts of different abstraction layers can be maintained.

Tailorable abstraction layer hierarchy

As the results of the study illustrate, the application of abstraction layers is not standardized in industry, is highly influenced by the application domain (e.g., automation, avionics, or healthcare), and varies depending on the engineering context (e.g., the specific system type, properties of the supplier-integrator relationship). In some cases, the use of abstraction layers is formally imposed by standards.

In summary, the study showed that the application of abstraction layers in industry depends largely on the engineering context and in particular on the responsible engineers' intuition and experience. Therefore, improved method guidance for specifying artifacts across different abstraction layers of an embedded system were needed at the beginning of the SPES project.

2.4 References

[OMG 2010a] Object Management Group: OMG Systems Modeling Language™ (OMG SysML) Language Specification v1.2. OMG Document Number: formal/2010-06-02.

[OMG 2010b] Object Management Group: OMG Unified Modeling Language™ (OMG UML), Infrastructure v2.3. OMG Document Number: formal/2010-05-03.

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal, Vol. 17, No.1, 2012, pp. 57-78.

Part II

The SPES Modeling Framework

Prof. Dr. Manfred Broy
Prof. Dr. Werner Damm
Dr. Stefan Henkler
Prof. Dr. Klaus Pohl
Andreas Vogelsang
Dr. Thorsten Weyer

3

Introduction to the SPES Modeling Framework

Today's and, even more so, the future development of embedded systems faces a variety of challenges. Key success factors to meeting these challenges are suitable concepts for abstraction and structure at different levels of granularity. The result of these concepts is a seamless development approach that heavily facilitates reuse and automation. A basic requirement for such a seamless approach is a clear notion of a system that is formalized by a comprehensive modeling theory. According to this modeling theory, a modeling framework has to provide appropriate models and description techniques for modeling the different aspects and artifacts of system development. This section explains these conclusions and introduces the idea of system and the modeling framework. It also references the modeling theories used in SPES.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_3,

3.1 Motivation for the SPES Modeling Framework

The aim of model-based development is to use models as main development artifacts in all phases of the development process. It promises to increase the productivity and the quality of the software development process by raising the level of abstraction at which the development is done, as well as the degree of automation, with the help of models that are tailored and appropriate for specific development tasks.

Current model-based approaches

Even though adopted in practical development of embedded systems today, model-based development approaches often fail due to the lack of sufficiently powerful modeling theories and missing integration of theories, methods, and tools. The models applied in the development process are based on separate and unrelated modeling theories (if foundations are given at all), which makes the transition from one model to another unclear and error-prone.

3.2 Characteristics of Software-Intensive Embedded Systems

Embedded systems and the SPES modeling framework

An embedded system can be characterized as a technical system that operates in a physical and technical environment and is built by means of technical resources that collaborate in order to achieve an overall purpose (see [Braun et al. 2010]). Embedded systems monitor and control their environment using variables that refer to specific properties of the environment (e.g., physical or technical properties; see [Parnas and Madey 1995]). IEEE Standard 1362 states that a system can be characterized as "software-intensive" if the software of the system is the major technical challenge and perhaps the major factor that affects its schedule, cost, and risk (see [IEEE 1362]). Typically, software-intensive embedded systems consist of software and hardware.

Software-intensive embedded systems are widespread in our daily life. They can be found in many application domains such as automation, healthcare, consumer electronics, avionics, transportation, and automotive.

Addressed characteristics of embedded systems

Software-intensive embedded systems exhibit some characteristics that have a far-reaching impact on the corresponding engineering and modeling approach with which they are developed:

- *Multifunctional:* Software-intensive systems provide a wide range of functionalities, i.e., they offer a variety of functions that interact with the environment and additionally with each other (see [Broy 2010]).
- *Complex:* A significant increase in the complexity of software-intensive embedded systems can be observed over the last years. This corresponds with the effect that perceivable functions are increasingly being realized by integrating fine-grained software-intensive embedded subsystems. Therefore, the complexity of such systems increases in two ways: the complexity of a single subsystem (intrasystem complexity) and the complexity of the relationship to other subsystems (intersystem complexity).
- *Reactive and interactive:* Software-intensive embedded systems are generally interactive or reactive systems. They are characterized by the constant interaction and synchronization between the system and its environment. While for interactive systems the interaction is determined by the system, the interaction of reactive systems is determined by the physical-technical environment.
- *Distributed:* In many cases, software-intensive embedded systems are no longer realized within a single electronic control unit (ECU) but distributed over a network of logical or physical components that interact with each other heavily in order to realize the desired functionality. These components execute their computations simultaneously on multiple cores in the same chip, with different threads on the same processor, or on physically distributed systems.
- *Control of continuous physical and technical processes:* Software-intensive embedded systems are frequently used to control physical processes and devices that exhibit a time-continuous behavior. The controller within the software-intensive embedded system is implemented in software and is consequently asynchronous and time-discrete. An engineering methodology for developing embedded systems must accommodate for that fact.
- *Exhibiting real-time properties:* Many software-intensive embedded systems must meet real-time requirements during system operation, for instance, to be able to guarantee the safety of the occupants within a vehicle. In such cases, the system must fulfill certain constraints that restrict the time behavior of the system's response if a specific crucial event in the environment of the system occurs.
- *Safety-critical:* Safety is a major quality of embedded systems that must be considered in any activity during engineering. Safety can be characterized as the extent to which the system under development will not have effects on its environment that result in harm to people,

significant monetary losses, or any other negative impacts to its environment.

3.3 The Principles of the SPES Modeling Framework

The overall requirements for the SPES engineering methodology as described in Chapter 2 led to a set of fundamental principles for the SPES modeling framework. These principles aimed at establishing specific ways of thinking to be applied when performing the modeling activities suggested by the engineering process for software-intensive embedded systems in order to meet the requirements from the application domains and the characteristics of such systems. These principles are:

- *Distinguishing between problem and solution:* This principle aims at distinguishing between the analysis of the underlying problem that has to be solved and the development of an appropriate solution in the form of a software-intensive embedded system.
- *Explicitly considering system decomposition:* Decomposition plays an important role as a lever to master complexity in nearly all engineering activities. It encompasses, for example, the decomposition of systems into subsystems, of functions into subfunctions, or the decomposition of hardware topologies. Following the decomposition of the system, its engineering process can be divided into a number of individual fine-grained engineering processes, complemented by certain activities to support the integration of the various engineering artifacts.
- *Seamless model-based engineering:* This principle aims at establishing a continuous model-based documentation or specification of all the information that is created during the different engineering activities. The notion “model-based” is used in two related ways. Firstly, the conceptual structures of the artifacts are defined by metamodels that specify the information structure of the artifact as well as the structural dependencies between artifacts. Secondly, the relevant information is documented or specified using conceptual modeling languages. In addition, an engineering approach can be characterized as “seamless” if relations between different types of artifacts exist and have clearly defined (formal) semantics. This makes it possible to use the models not only as documentation but also to perform automatic analysis and to transform one model into another.

- *Distinguishing between logical and technical solutions:* This principle aims at separating the logical solution, which focuses on general solution concepts and corresponding conceptual properties of the system under development, from technological constraints and technological design decisions. By following this principle, the engineers can clearly separate the logical solution, which is largely independent of technological constraints and technology-related design decisions, from the actual technical solution for the system under development. Due to the fact that the logical solution is largely independent of technological design decisions, the logical solution is more stable than the technical solution and can be reused for different technical realizations.
- *Continuous engineering of crosscutting system properties:* This principle aims at establishing the ability to consider crosscutting properties of the system under development. Typical crosscutting properties are safety or real-time properties of the system: they must be considered in any engineering activity and the corresponding artifacts, such as requirements, design, and implementation artifacts.

3.4 Core Concepts of the SPES Modeling Framework

To establish the fundamental principles mentioned above, the SPES modeling framework uses the following core concepts.

3.4.1 Abstraction Layers

Advantages of using abstraction layers

A system under development or a design element (e.g., a logical component) can be modeled on different abstraction layers. Less abstract models belonging to a lower abstraction layer may increase the level of detail of the description of the design element at hand. For example, the interface description may be refined as well as the behavior demanded. Therefore, increasing the level of detail, and at the same time decreasing the layer of abstraction, adds knowledge about the design element. Often, a reduction in the level of abstraction is accompanied by a refined granularity — that is, structurally significant design elements are decomposed into multiple finer grained items in order to keep them at a manageable size. Refining the granularity is, however, not a necessary condition when introducing a lower layer of abstraction. For example, it is possible to describe the same design item on two different layers of abstraction with the same granularity, but specify a certain aspect more

precisely. Despite an increasing level of detail, another motivation for introducing a dedicated abstraction layer can be the handover of an initial system specification to another organizational unit. In this way, the initial specification remains unchanged and the design element is refined on a new abstraction layer. Mappings between engineering artifacts allow these refinements to be traced. While models on lower abstraction layers provide more detail, the design elements still have to respect the aspects specified for their higher level counterparts.

3.4.2 Views and Viewpoints

Views and viewpoints according to [IEEE 1471]

Multiple stakeholders with different concerns are involved in the engineering process for a software-intensive embedded system. The aim of the concept of viewpoints is to separate the various concerns of different stakeholders during the engineering process. It serves as a construct for managing the different artifacts during the engineering process. IEEE Standard 1471 [IEEE 1471] characterizes viewpoints as a specification of the conventions for constructing and using a view. In other words, a viewpoint is a pattern or template that can be used to develop individual views on a system (and its environment). Typically, the specification of a viewpoint defines that viewpoint in terms of its syntax, semantics, and pragmatics by providing, among other things, the name of the viewpoint, the corresponding stakeholder concerns, the viewpoint language (probably given by a metamodel), and techniques that can be used during the construction and analysis of the corresponding view (see [IEEE 1471]). Given a viewpoint specification, a view can be characterized as a concrete model of the system that represents the information that is relevant for the corresponding viewpoint concerns by using the conceptual structure of the underlying viewpoint language.

3.5 The SPES Modeling Framework

While software-intensive embedded systems are becoming more and more complex, market pressure requires companies to develop high-quality products in a short time. To deal with such complexity, software and systems engineering approaches propagate a structured, well-defined design process consisting of several steps based on abstraction and refinement techniques (e.g., [Sage and Rouse 2009, Sommerville 2010]).

Structuring the problem space

Abstraction allows the designer to concentrate on the essentials of a problem. Refinement adds detail to abstract models while preserving

properties established on the more abstract level. In other words, a more concrete description of a design entity also has to fulfill all requirements of the abstract design entity it has been derived from (see Section 3.4.1).

Following the principle of separation of concerns [Dijkstra 1976, Tarr et al. 1999], the concept of viewpoints that provide artifacts and rules for describing different views of the system under development should be supported. A view is created by a set of models that describe the system under development and/or its parts, and is related to a viewpoint (see Section 3.4.2).

Structuring the solution space

In a complex design process, it is important to have a clear idea of decomposition in order to be able to ensure that the final implementation meets the requirements. A decomposition relationship within the SPES modeling framework introduces a level of interconnected subparts whose collaboration shall provide any functionality the decomposed unit shall provide. Engineering activities for the single parts should be largely independent of each other: firstly, to simplify work by limiting the required focus of attention, and secondly to enable development in a distributed fashion. Another point to note is that parts can be exchanged consistently provided that the part's original specification is also fulfilled by the new part (see Section 3.3).

While many modeling approaches partially support such concepts, they often do not cover the whole engineering space from initial requirements down to a final implementation. This means, for example, that these approaches cannot continuously consider crosscutting system properties (see Section 3.3) as they do not support traceability in the design process. Furthermore, none of these approaches consider a well-defined combination of abstraction layers and viewpoints at all. It is the aim of this work to provide a modeling framework that does not suffer from these shortcomings.

Using abstraction layers and viewpoints in the SPES modeling framework

In our approach, *abstraction layers* and *viewpoints* form a two-dimensional engineering space (see Fig. 3-1). Based on well-understood software engineering approaches, the SPES modeling framework focuses on the following viewpoints to support views, starting with solution-neutral requirements through to concrete technical solutions: *requirements* (see Section 3.5.1), *functional* (see Section 3.5.2), *logical* (see Section 3.5.3), and *technical* (see Section 3.5.4). Note that our approach is not limited to these viewpoints in principle. For example, in other application scenarios a geometrical viewpoint is required [Baumgart et al. 2011]. The requirements, functional, and logical viewpoints are especially pertinent in software engineering, also for systems with no technical background. The need to also support a technical viewpoint is driven by the fact that the SPES modeling

framework considers software-intensive embedded systems in which the software is affected by the physical environment of the system and has to interact with (or react to) the surrounding technical system (see Section 3.2).

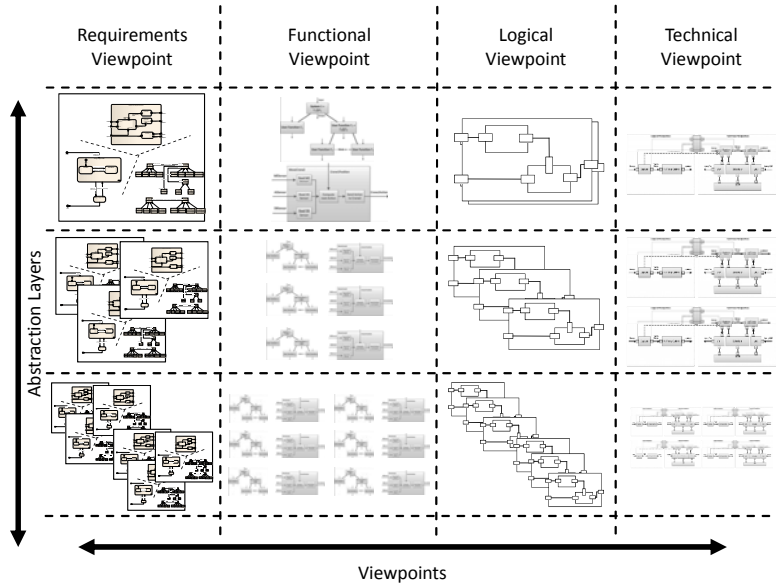


Fig. 3-1 *SPES Modeling Framework*

While the four viewpoints mentioned in the SPES modeling framework are the same for the different domains, the abstraction layers differ [Baumgart et al. 2010, Baumgart et al. 2011, Sikora et al. 2012]. For example, in the avionics domain, the abstraction layers *aircraft*, *system*, *subsystem*, *sub-subsystem*, *component,* and *unit* can be found, and in the automotive domain, the layers *supersystem*, *system*, *subsystem*, and *hardware/software component*. Hence, our approach abstracts from these different layers by supporting user-defined layers.

The SPES modeling framework is designed to be independent of any application domain (e.g., automation, automotive, avionics, energy, and medical). It defines fundamental concepts and how these concepts are related to one another. Domain-specific metamodels are interpreted in these concepts, thereby making general analysis techniques available for the specific application domain. In the following, we provide an overview of the viewpoints in Sections 3.5.1 to 3.5.4. In Section 3.5.5, we consider the relationship between the viewpoints that allows the seamless integration of the viewpoints into a comprehensive modeling approach.

3.5.1 Requirements Viewpoint

From system context to solution-oriented requirements

The goal of the requirements viewpoint is to support the requirements engineering process in a development project in eliciting, documenting, negotiating, validating, and managing requirements for the system under development. These requirements are derived from the system's context. The context of the system comprises entities from within the environment of the system, such as users, stakeholders, and external systems, but also legal documents. It also comprises physical properties of the environment that affect the system or are affected by the system in some way. Since these context entities are hence related in specific ways to the system under development, they must also be considered when eliciting requirements. The requirements viewpoint provides a requirements artifact model that allows for a systematic consideration of the context, the entities therein, and the resulting requirements. The basic aim of the artifact model of the requirements viewpoint is to capture the requirements from the system's context completely and correctly and to provide a means for structuring these artifacts on different levels of abstraction (cf. Section 3.4.1).

Requirements artifacts

The artifact model comprises a number of artifact types that are briefly explained in the following:

- *The context model* regards the system as a black box and documents the context of the system under development. It provides the basis for systematically eliciting the requirements that the system under development must satisfy during operation in order to meet its overall purpose. Context models are well suited for use as a foundation when performing activities for validating the correctness of requirements (see [Weyer 2011]).
- *The goal model* documents the stakeholders' goals with regard to the system under development (see [Levenson 2000, Lamsweerde 2009]). Goals can be elicited by analyzing the system's interaction with entities in the context and they serve as a rationale for more concrete requirements.
- *The scenario model* documents examples of concrete interactions between the system under development and its context. Each scenario describes an example in which at least one goal is satisfied. Scenarios can also be used to elicit new system goals [Potts 1995, Yu 1997].
- *The solution-oriented requirements model* documents the concrete and complete technical requirements that the system under development has to realize in order to satisfy its purpose during operation. These requirements must be as precise as possible to serve

as a foundation for the realization of the system under development. For solution-oriented requirements models, the SPES modeling framework distinguishes between three complementary model types (see [Davis 1993]): the structural requirements model documents requirements that describe the structure of the information that is exchanged between the system and its environment; the behavioral requirements model describes the externally visible behavior of the system by documenting the externally recognizable state space and corresponding state transitions; the operational requirements model documents required system functions by considering the functional relation between incoming and outgoing flows of information as well as the necessary control flow.

The requirements viewpoint is explained in detail in Chapter 4.

3.5.2 Functional Viewpoint

The purpose of a system is to offer a set of user functions [Broy et al. 2007]. Typical systems offer a number of different user functions and each user function serves a specific purpose. A system function is characterized by a particular observable system behavior in terms of specific interactions between inputs to the system (e.g., via sensors or user actions) and outputs of the system (specific effects on actuators, general reactions). The observations are captured in terms of the primary events of the user function (see [Broy 2010]).

User functions vs. realization functions

An example of a user function in a car might be an adaptive cruise control (ACC) or a cooling control for the engine. In both cases, the behavior of the user functions can be defined via the inputs of the system from the environment and the outputs of the system to the environment. Thus, both user functions are user functions of the system *Car*. All user functions of the system *Car* are structured in the functional black box model within the functional viewpoint. In contrast to this notion, a functionality that is needed to fulfill the user function is not considered as a user function of the system *Car*. Let us assume that the ACC user function is realized such that at one stage, the input of different speed sensors has to be aggregated. We could consider *SensorAggregation* as a user function of the system *Car* as well. However, this is not our understanding of a user function in the SPES modeling framework. Instead, *SensorAggregation* is considered part of a description of an abstract realization of the system *Car* (cf. Section 3.5.3). These parts are called functions (in contrast to user functions) and are specified in the functional white box model of the functional viewpoint. As the name suggests, in the white box model, user functions of the black box model

are described by an abstract description of their realization. This differentiation will become clearer within the following sections about the functional viewpoint and its relation to the logical viewpoint.

The functional viewpoint integrates the set of user functions into a comprehensive model of the system functionality. This functional model describes the system behavior as it is observed at the system boundary. In contrast to the requirements viewpoint, where requirements are captured with respect to a certain usage context and at a certain level of granularity, the models of the functional viewpoint integrate these requirements into a comprehensive system specification. This especially includes behavior that arises from the complex interplay of different user functions.

User functions

We define the notion "user function" as a concept that has a specific purpose and corresponds to a determined behavior in the form of an interaction across the system boundaries. In addition, user functions typically have identifying names. The behavior of a user function can be captured by a behavior specification.

If there are dependencies between user functions such that the output of a user function depends implicitly on the behavior of another user function, this is referred to as a functional dependency or feature interaction and we model this using *modes*. This will be discussed in Chapter 6.

Hierarchies of user functions

According to this idea, we describe the functionality of a system under development using functional hierarchies in which we combine user functions into functional groups. The leaves of the resulting hierarchical structure correspond to individual atomic user functions. A functional hierarchy specifies the functionality of the system under development at a specific level of abstraction. The granularity that is chosen for a function hierarchy depends on the choice and the skill of the developer. The more intelligently the functional hierarchy is chosen, the more independently the user functions can be described, and the clearer the functional dependencies between the user functions captured by the modes.

3.5.3 Logical Viewpoint

In order to realize the desired functionality that is specified in the models of the functional viewpoint, the developer has to think about a decomposition of the system under development into an architecture of logical components. The logical viewpoint describes this glass box structural decomposition of the system, whereas the functional black box model of the functional viewpoint in particular focuses purely on

describing the black box behavior (see [Schätz 2005]). The result is a description of the logical solution independent from any technological constraints. This description can be reused for multiple platforms. The reasons for decomposing the system under development into subsystems are manifold. In addition to mastering the complexity, further aims of the logical viewpoint are the division of labor and in particular, improving the capability of reuse. Grouping functionality that contributes to the realization of multiple user functions into one subsystem can save development costs and increase quality (see [Lim 2002]).

Logical component architecture

A logical component architecture as described in the logical viewpoint consists of a number of logical components that are connected via logical channels. Logical components exchange data via their logical channels, in the sense of a data flow architecture. Logical architectures can be structured hierarchically such that coarse-grained logical components are themselves again broken down into fine-grained logical components. At the level at which subsystems should not be further broken down, they can in turn be described by behavior description techniques such as state machines with input and output. The decomposition of a system into logical components is a starting point for the next iteration of eliciting requirements and defining user functions for each logical component.

The behavior of individual logical components can—similar to user functions—be represented by behavior descriptions (e.g., state machines). These behavior descriptions implement the individual logical components and their logical behavior. If we manage to capture all of these logical components in their logical behavior using state machines, a purely logical system simulation can be performed.

Relationship between user functions and logical components

An important question is how the relationship between the user functions and the logical components of the logical component architecture can be used methodically. Typically, only a portion of the logical component architecture is needed to provide a specific user function. We can thus define micro-architectures that show the portion of the logical component architecture that is relevant for a user function. This is interesting insofar as it may ultimately determine which of the logical components are involved in the provision of a user function. Particularly appealing is the representation of the modes from the hierarchy to the level of logical structure and the function of their technical components.

3.5.4 Technical Viewpoint

The technical viewpoint combines the software of the system under development with its hardware. It thus contains a description of the physical architecture. A deployment mapping specifies where software tasks of the logical viewpoint are executed and where and how logical subsystems are realized. The viewpoint considers aspects such as timing, resource consumption, and redundancy insofar as these have not been addressed before.

Goals

The main goals of the technical viewpoint are:

- Providing a description of the target hardware, with ECUs, memory, communication infrastructure, and peripheral devices
- Fixing the deployment of software modules
- Realizing logical subsystems
- Studying the interaction of software and hardware
- Ensuring that the behavior conforms to the specifications of the logical viewpoint, and that constraints concerning timing, independency, etc. are observed

The technical architecture comprises components for information processing (including communication) and their connection to the environment via sensors and actuators. The nature of the controlled system may have a considerable impact on the structure of the architecture and the characteristics of the information-processing components. These components will usually be described in the form of models abstracting from the details of the actual components that are used. The models also cover relevant services of operating systems, middleware, and so on. There are elaborated domain-specific approaches such as AUTOSAR (automotive) or IMA (avionics) that will often be employed.

The realization of the logical viewpoint via the deployment mapping results in a description of the system close to its final implementation. Therefore, properties that have been specified abstractly in previous engineering steps must be shown to have been realized in the technical architecture. Most prominently, these concern resource consumption, timeliness, and issues such as reliability and availability. For instance, tasks may be regarded as independently executable in the models of the logical viewpoint. However, if they are allocated on the same computing resource, they now have to be scheduled in a way that is consistent with all requirements. Communication, which was modeled as point-to-point and without delay in the logical viewpoint, has to be shown as appropriately realized by shared media such as busses.

Thus, the technical viewpoint studies the properties of the final implementation and has to establish that the physical realization meets all logical requirements.

3.5.5 Relation between Viewpoints

Relating the requirements and functional viewpoints

The requirements viewpoint introduces context factors, such as external stimuli or usage factors imposed by users or other systems, into the development process and establishes them as part of the requirements of the system under development. It therefore serves as a starting point for other viewpoints in the SPES modeling approach, as it specifies requirements that must be adhered to by other viewpoints. For example, solution-oriented requirements models of the requirements viewpoint must be fulfilled by user functions that are integrated in the functional hierarchy developed in the functional viewpoint. Furthermore, the external interfaces specified in the context models must correspond to logical (external) interfaces in the functional viewpoint as well as in the logical viewpoint. However, this is not a strict top-down process. Subsequent development activities in other viewpoints may require artifacts from the requirements viewpoint to be altered, modified, removed, or extended. Furthermore, the system context may also directly affect viewpoints other than the requirements viewpoint. For example, if the system to be developed has to be integrated into an existing environment consisting of legacy systems, these systems will inevitably affect the technical viewpoint.

Relating the functional and logical viewpoints

The logical viewpoint describes the internal logical structure of the system by means of communicating logical components. Thus, the viewpoint considers the system in a glass box view in contrast to the functional viewpoint where the system is considered as a black box. The relation between a user function (from the functional viewpoint) and a logical component (from the logical viewpoint) is often mixed up. A user function formalizes a part of the requirements that a system must fulfill at its boundary, whereas a logical component captures a part of functionality that lies within a system and that contributes (among other things) to the realization of a user function. In general, there is an n:m mapping between user functions and logical components. This means that one user function can be realized by a number of logical components, and one logical component can contribute to the realization of a number of user functions.

Fig. 3-2 illustrates the relation between these two notions. As shown, the system under development S is structured into a hierarchy of user functions (functional black box model). For each user function, there is a

functional white box model. The figure only shows the functional white box model for user function UF_1. The functions of the white box model are allocated to logical components in the logical viewpoint. The logical component LC_1 itself can in turn also be considered a system under development in the next lower abstraction layer. We can again provide a functional black box model (with function F_1 as one of the user functions), functional white box models, and a logical component architecture for the SUD LC_1.

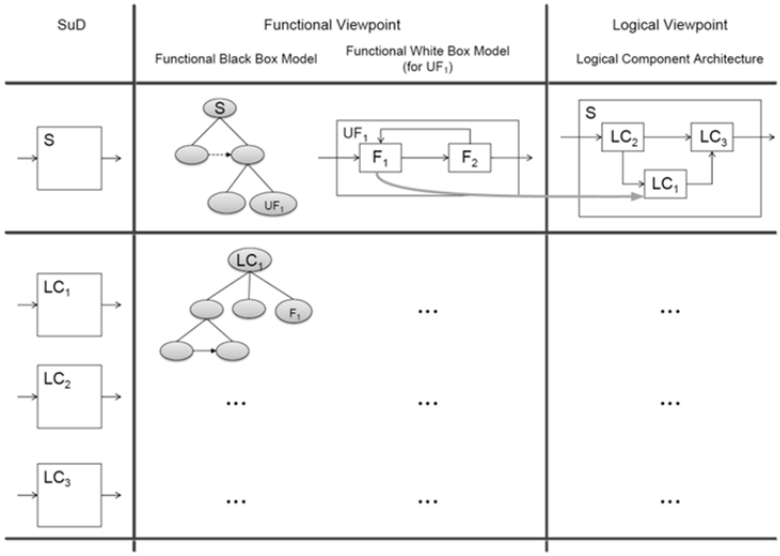


Fig. 3-2 *Relation between the functional and logical viewpoints across abstraction layers*

Relating the logical and technical viewpoints

At the core of the relation between the logical and the technical viewpoints is the deployment mapping. It specifies where and how logical components are realized on the technical architecture: which technical parts (ECUs, busses etc.) are involved in implementing such logical components, which communication resources are used in their interaction, and so on. Once the deployment is specified, a check is required to determine whether properties established for the logical viewpoint remain valid in the technical viewpoint. A typical issue that arises is schedulability, for instance, when several software tasks have been allocated to one ECU. The availability of features of the target has a considerable impact on the form of the deployment mapping and the type of analyses to be performed. Platforms such as AUTOSAR provide an abstraction layer that alleviates several of these tasks. Another issue in this design step concerns requirements inherited from previous steps that

now have to be implemented. For example, redundancy, reliability, and independence usually have to be taken care of when moving to the technical viewpoint.

3.6 Underlying Modeling Theories

The SPES modeling framework is founded on several formal modeling theories and uses these theories' basic concepts in a continuous modeling approach. However, the SPES modeling framework was not based on one single specific theory and may be formalized using any of the underlying modeling theories. The aim of this was to allow practitioners to tailor the SPES modeling approach for different development projects and to allow for a maximum of flexibility in formalization. The underlying modeling theories are briefly summarized below.

The SPES modeling framework supports the use of formalized notions but does not force it. In the requirements viewpoint, for example, to document scenarios, informal sequence charts can be used in addition to formal message sequence charts [ITU 2004]. While informal sequence charts are mainly used for describing and discussing the system's functions and behavior with stakeholders, formal message sequence charts may be used to specify strict and formal solution-oriented requirements consistent with different components.

Similarly, in order to integrate different architecture viewpoints (functional, logical and technical), formal semantics can be used. Therefore, at least two different formal modeling theories have been developed that fit the SPES modeling approach. One modeling theory describes modeling entities such as functions, subsystems, or technical components using stream processing functions. In this theory, a stream processing function describes the behavior of an entity at its interface in terms of input and output data streams (see [Broy 2010]). Another modeling theory describes modeling entities as heterogeneous rich components [Damm et al. 2005], where the term "rich" alludes to the key ingredient of heterogeneous rich components to provide (rigorous) interface specifications for multiple aspects, encompassing both functional and extrafunctional (e.g., safety and real-time) characteristics of components. Specifically, the proposal is to use contract-based specifications that allow, for each aspect, characterization of the allowed design context of a component.

Besides the different formal modeling theories that can be chosen electively, there is a common understanding that forms the foundation of the complete SPES modeling framework with all of its views. The SPES

modeling framework distinguishes between the system under development and the system's context. Whereas modeling the context is a key element of the requirements viewpoint, the different architectural views focus on modeling the interfaces connecting the system with its context. The system may be decomposed into subsystems or components with their own contexts. The context of a subsystem, for example, will contain the context of the overall system as well as the other subsystems of the system.

3.7 Overview of the Following Chapters

All four viewpoints of the SPES modeling framework are explained in more detail in the following chapters. Chapter 4 outlines the requirements viewpoint and illustrates how requirements from the context can be documented and refined by means of requirements models. The functional viewpoint is explained in Chapter 5 and shows how the requirements models from the requirements viewpoint can be refined into a functional architecture of the system that fulfills the solution-oriented requirements. Chapter 6 describes the logical viewpoint that allows specification of a logical architecture that consists of internal and external interfaces of the system in compliance with the system requirements as well as the functional architecture. Finally, Chapter 7 illustrates the technical viewpoint. This viewpoint maps a logical architecture onto concrete physical and technical components such that the system requirements are fulfilled.

Safety and real-time are two very important concerns of embedded system development. These concerns are crosscutting and have to be considered in every viewpoint on every abstraction layer that is used during development. Chapters 8 and 9 show how safety and real-time respectively are accounted for during the development within all viewpoints.

All four viewpoint chapters and both crosscutting concerns illustrate their respective concepts by means of a common example system in the form of a case study. The case study is a cylinder head production system taken from the automation domain. The cylinder head production system is an assembly line that produces cylinder heads for gasoline engines. The system takes raw material that is fed into the production cell. It consists of an input and an output conveyor belt, a milling station, a grinding station, a measuring station, as well as an assembly station that finally delivers the finished product. Production of cylinder heads must obey strict real-time constraints, as material may only be processed under

the right physical conditions. Furthermore, system safety is an important concern that must be observed.

3.8 References

[Baumgart et al. 2010] A. Baumgart, P. Reinkemeier, A. Rettberg, I. Stierand, E. Thaden, R. Weber: A model-based design methodology with contracts to enhance the development process of safety-critical systems. In: Proceedings of the 8th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems, SEUS'10, 2010, pp. 59-70.

[Baumgart et al. 2011] A. Baumgart, E. Böde, M. Büker, W. Damm, G. Ehmen, T. Gezgin, S. Henkler, H. Hungar, B. Josko, M. Oertel, T. Peikenkamp, P. Reinkemeier, I. Stierand, R. Weber: Architecture modeling. In: OFFIS Technical Report, OFFIS Oldenburg, March 2011.

[Braun et al. 2010] P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Müller, B. Penzenstadler, K. Pohl, T. Weyer: Guiding requirements engineering for software-intensive embedded systems in the automotive industry. Computer Science - Research and Development. DOI: 10.1007/s00450-010-0136-y, 2010.

[Broy 2010] M. Broy: Multifunctional Software Systems: Structured modelling and specification of functional requirements. In: Science of Computer Programming, Vol. 75, No. 12, 2010, pp. 1193-1214.

[Broy et al. 2007] M. Broy, I. Krüger, M. Meisinger: A formal model of services. In: ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 16, No. 1, ACM, New York, 2007.

[Damm et al. 2005] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, E. Böde: Boosting re-use of embedded automotive applications through rich components. In: Foundations of Interface Technologies, FIT'05, 2005.

[Davis 1993] A. M. Davis: Software Requirements – Objects, Functions, States. 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Dijkstra 1976] E. Dijkstra: A discipline of programming. Prentice-Hall Series in Automatic Computation, 1976.

[IEEE 1362] Institute of Electric and Electronic Engineers: Guide for information technology – system definition – concepts of operations (IEEE 1362-1998), IEEE Press, 1998.

[IEEE 1471] Institute of Electric and Electronic Engineers: Architectural Description of Software-Intensive Systems (IEEE 1471-2000), IEEE Press, 2000.

[ITU 2004] International Telecommunication Union: ITU-T Z.120: Message Sequence Chart (MSC), 2004.

[Lamsweerde 2009] A. van Lamsweerde: Requirements Engineering – From System Goals to UML Models to Software Specifications. Wiley, West Sussex, 2009.

[Levenson 2000] N. Levenson: Intent Specifications – An approach to building human-centered specifications. IEEE Transactions on Software Engineering, Vol. 26, No. 1, 2000, pp. 15-35.

[Lim 2002] W. Lim: Effects of reuse on quality, productivity, and economics. IEEE Software 11(5), 2002.

[Parnas and Madey 1995] D. L. Parnas, J. Madey: Functional documents for computer systems. In: Science of Computer Programming, Vol. 25, No. 1, pp. 41-61.

[Pohl 2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, Berlin/Heidelberg, 2010.

[Potts 1995] C. Potts: Using schematic scenarios to understand user needs. In: Proceedings of the ACM Symposium on Designing Interactive Systems – Processes, Practices, Methods and Techniques (DIS'95). ACM, New York, 1995, pp. 247-266.

[Sage and Rouse 2009] A. P. Sage, W. B. Rouse: Handbook of Systems Engineering and Management. John Wiley and Sons, 2nd Edition, 2009.

[Schätz 2005] B. Schätz. Building components from functions. In: Electronic Notes in Theoretical Computer Science, Vol. 160. Proceedings of the International Workshop on Formal Aspects of Component Software FACS 2005.

[Sikora et al. 2010] E. Sikora, M. Daun, K. Pohl: Supporting the consistent specification of scenarios across multiple sbstraction levels. In: R. Wieringa, A. Persson (Hrsg.): Proceesdings of the 16th Intl. Working Conf. on Requirements Engineering: Foundation for Software Quality. LNCS 6182, Springer, Berlin/Heidelberg, 2010, pp. 45-59.

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal, Vol. 17, No.1, 2012, pp. 57-78.

[Sommerville 2010] I. Sommerville: Software Engineering. Pearson, 9th Edition, 2010.

[Tarr et al. 1999] P. Tarr, H. Ossher, W. Harrison, Jr. S. M. Sutton: N degrees of separation: multi-dimensional separation of concerns. In: ICSE '99: Proceedings of the 21st international conference on Software engineering, ACM, New York, 1999, pp. 107-119.

[Weyer 2011] T. Weyer: Kohärenzprüfung von Anforderungsspezifikationen: Ein Ansatz zur Prüfung der Kohärenz von Verhaltensspezifikationen gegen Eigenschaften des operationellen Kontexts. Südwestdeutscher Verlag für Hochschulschriften, 2011.

[Yu 1997] E. Yu: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), IEEE Computer Society Press, Los Alamitos, 1997, pp. 226-235.

Marian Daun
Bastian Tenbergen
Dr. Thorsten Weyer

4

Requirements Viewpoint

The requirements viewpoint defines concepts and techniques for systematically eliciting and specifying the requirements for a system under development. The requirements viewpoint differentiates between different artifact types that document different information elicited during requirements engineering:

- *Context, which documents the operational environment in which the system under development is embedded*
- *Goals, which document stakeholder intentions with regard to the system under development*
- *Scenarios, which document typical interactions between the system under development and its context*
- *Solution-oriented requirements, which document the requirements for the system under development in a precise and complete manner*

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_4,

4.1 Introduction to the Requirements Viewpoint

Separation of requirements and solution

The requirements viewpoint comprises the part of the SPES modeling framework that primarily deals with the accurate, complete, and consistent specification of system requirements. These requirements serve as input for functional analyses and architecture design (see Chapters 5 to 7). The goal of the requirements viewpoint is to:

- Gain a comprehensive understanding of the system under development
- Foster the best possible freedom in development by preventing premature commitment to possible solutions
- Supply the necessary information such that decisions pertaining to concrete implementation can be made during subsequent architecture design

Solution-neutral and solution-oriented requirements; co-design process

In the requirements viewpoint, a strict separation between stakeholder intentions and solution-oriented requirements is maintained. In order to support this separation, the requirements viewpoint contains three solution concepts:

- *Solution-neutral requirements* describe the intentions of the stakeholders and the added benefit that can be gained for the stakeholders [Leveson 2000]. Concrete aspects of a possible solution are ignored.
- *Solution-oriented requirements* describe necessary properties of operations, system states, and the information structure, as well as qualities that a solution must possess [Pohl 2010]. Solution-oriented requirements are the connection between solution-neutral requirements and concrete implementations.
- The *intertwined development* of requirements artifacts is based on a goal-/scenario-oriented, step-by-step refinement of requirements from solution-neutral to solution-oriented requirements. Due to the step-by-step, artifact-based refinement, the intertwined development allows for traceability between requirements artifacts, ensures requirements consistency between the artifacts, and leads to completeness with regard to the requirements artifacts and the requirements specification.

Artifact model, types of requirements models

The requirements viewpoint documents a complete system requirements specification by means of partial diagrams. Therefore, each artifact model contains a number of different requirements diagrams (see Section

4.2). Due to the number of different requirements artifacts and their interrelations, the requirements viewpoint is very comprehensive as well as complex. Therefore, different model types are used. By using different model types within the requirements viewpoint, the corresponding view is constructed by integrating each model type based on common facts. Typical model types are goals and scenarios, as well as structural, operational, and behavioral models (see [Pohl 2010] and Section 4.2.4).

The artifact model of the requirements viewpoint is explained below (Section 4.2). In addition, the integration of the requirements viewpoint with other viewpoints and abstraction layers of the SPES modeling framework is illustrated in Section 4.3. Finally, we outline a requirements engineering process across several abstraction layers: it can be used to systematically develop requirements of the system under development (SUD) and can be tailored for individual project needs (Section 4.4).

4.2 Requirements Artifacts

In this section, we briefly outline the artifact model of the requirements viewpoint. Each subsection outlines one artifact type and gives a short example.

4.2.1 Context Model

Importance of the system context

The context of the system is that part of the operational infrastructure that does not belong to the system (and therefore cannot be influenced during development) but surrounds the system once it has been deployed (and therefore strongly impacts the definition of requirements for the SUD). If the context of the SUD is not properly understood, it is impossible to properly define and interpret the requirements for the SUD [McMenamin and Palmer 1984, Davis 1993, Jarke and Pohl 1994, Hammond et al. 2001]. The requirements viewpoint therefore contains context models for modeling that part of the environment that influences the system. Context models can be used to document constraints from the physical environment of the system that limit the scope, solution space, or development process (e.g., the environment it will be deployed in, company-specific regulations, or laws and legislation that must be adhered to).

Context entities, interfaces, and system interaction

Context models focus on the system's desired interaction with its environment or, more precisely, its context entities [Weyer 2011]. Context entities are, for example, external actors, sensors, and other

systems in the environment that interact with the SUD. Each specified context entity must be present in at least one scenario model (see Section 4.2.3) so that the SUD's interaction with each entity can be assessed. Context models allow system goals to be determined and give a first impression about the SUD's interaction with its context. In the SPES modeling framework, context models also define the interfaces of the functional black box model in the functional viewpoint (see Section 5). Further information on context models can be found in [Weyer 2011]. In the requirements viewpoint of the SPES modeling framework, a number of different types of context models can be used. For example, structural diagrams such as SysML block definition diagrams or internal block diagrams [OMG 2010a] can be used to document static/structural context information. On the other hand, dynamic aspects of the context can be documented using Petri nets [Reisig 1991] or communicating finite state machines [Lynch and Tuttle 1989, Alfaro and Henziger 2001].

Example of a context model

Fig. 4-1 shows an example of the context model of a simplified automation system as a SysML block definition diagram. The SUD (the <<block>> stereotype in the middle) is treated as a black box, that is, no internal properties are considered. There are a number of context entities (<<actor>> stereotypes) that communicate with the SUD, either receiving output from the SUD or producing input to the SUD.

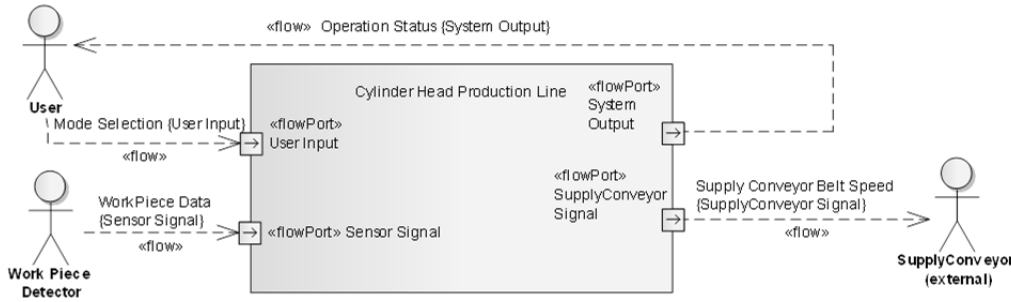


Fig. 4-1 *Example of a context diagram* [1]

[1] All figures in the requirements viewpoint have been modeled using Enterprise Architect®

Hint 4-1 lists the rules that have been defined in the requirements view for ensuring that the context of the SUD has been modeled completely and correctly.

Rules for checking context models

Hint 4-1*: Rules for checking context models*

- ❑ Have all actors in the system context that receive output from or produce input to the SUD been considered?
- ❑ Have all inputs that the SUD receives from the environment or from entities within the system context been considered?
- ❑ Have all outputs that the SUD delivers to the system context or to context entities been considered?

4.2.2 Goal Model

Capturing stakeholder intentions

Goal models document the intentions of stakeholders when they are conceiving the system. They represent a first manifestation of the stakeholders' system vision. Goals give rationales and justifications for the functionalities and features the system must possess. Goals ignore concrete aspects of the solution and hence serve as an essential means for negotiating requirements and their necessity with regard to the system envisioned. The purpose of negotiating requirements on the basis of goals is to establish a common understanding of the envisioned among all stakeholders. In addition, goals can be used to document necessary quality aspects such as the system's safety features (see Chapter 8) or real-time behavior (see Chapter 9) that in turn will be specified using solution-oriented requirements (see Section 4.2.4).

Goal types and goal refinement

In goal models, relationships can be identified between goals, functions, and qualities. For example, goals might be in direct conflict with one another (i.e., fulfilling one goal will make it impossible to fulfill a conflicting goal), or the fulfillment of goals may contribute positively or negatively to the fulfillment of another goal (i.e., make it easier or harder to achieve the other goal). In addition, goals can be refined using AND and OR refinements: AND refinements denote that a number of refining goals have to be fulfilled in order to fulfill the refined goal; OR refinements denote that at least one of the refining goals has to be fulfilled in order to fulfill the refined goal. Furthermore, we can distinguish between hard and soft goals. Hard goals are goals whose fulfillment can be verified by means of simple yes/no checks (i.e., either the goal has been fulfilled or not). In contrast, soft goals represent goals that the system to be developed fulfills to a certain degree.

Sources of goals/stakeholder intentions

Goals can be determined in part from the context model but also through stakeholder collaboration. Typical stakeholders who may contribute goals to a system are clients, contractors, product managers, business managers, technical leaders, certifiers or certifying authorities, or the legislative authority. Goals and goal modeling are explained in more detail in [Yu 1997, Lamsweerde 2009, Pohl 2010]. In the requirements view of the SPES methodology, KAOS goal diagrams, i* models, or SysML requirements diagrams can be used to model this artifact type.

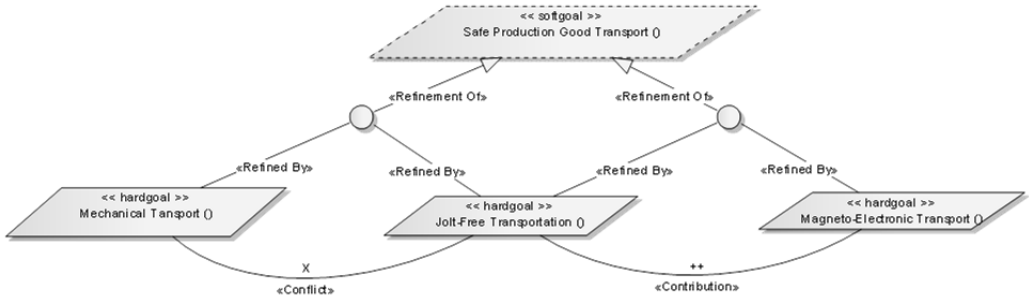


Fig. 4-2 *Example of a goal diagram*

Example of a goal diagram

Fig. 4-2 shows the goal diagram of an example system using the KAOS notation [Lamsweerde 2009]. The goals are structured hierarchically through AND and OR refinement. In this diagram, the top-most soft goal is refined by means of two alternatives (OR refinement). One alternative consists of two hard goals that both have to be fulfilled for the soft goal to be fulfilled (AND refinement). However, there is a conflict between these two goals, which may indicate that this alternative is not a suitable refinement of the soft goal. The other alternative also consists of two hard goals (one of which is also a refinement of the other alternative) that both have to be fulfilled for this alternative to be a valid refinement of the soft goal (AND refinement). The diagram shows a contribution link between the two goals in this alternative, indicating that the fulfillment of one goal positively contributes to the fulfillment of the other goal. Hence, this alternative is preferable over the other alternative.

Hint 4-2 lists the rules that have been defined in the requirements viewpoint for ensuring that all goals for the SUD have been modeled completely and correctly.

Hint 4-2: *Rules for checking goal models*

- ❑ Have all hard goals of stakeholders been captured?
- ❑ Have all soft goals of stakeholders been captured?
- ❑ Have all abstract (hard or soft) goals been refined using AND and OR refinements?
- ❑ Have all positive and negative contributions from one goal to another goal been uncovered and documented?

Rules for checking goal models

4.2.3 Scenario Models

Example interactions with the system

Scenarios specify example interactions of the system with its context. They allow requirements to be determined by modeling the system's interaction with context entities that have been identified in the context models (see Section 4.2.1). This enables the system's benefit and impact on the system context to be assessed. The actors that are present in any scenario model must be present in at least one context model that has been specified earlier. Scenarios fulfill the goals that have been specified in the goal models (see Section 4.2.2). In the requirements viewpoint, any goal has to be fulfilled by at least one scenario and every scenario must fulfill at least one goal. Scenario execution is typically constrained by preconditions. After scenario execution, specific postconditions must hold for the entire system. Furthermore, scenarios may specify some internal states that can be used to draft an initial specification of the behavioral requirements models of solution-oriented requirements (see Section 4.2.4). In scenario models, similarly to the goal models, the system is considered as a black box. Hence, there must not be any indication within either model that depicts the internal structure of the SUD. We can distinguish between different types of scenarios, for example:

- ❑ *Main scenarios:* Main scenarios describe the standard way of fulfilling one or more goals.
- ❑ *Alternative scenarios:* Alternative scenarios describe alternative ways of fulfilling the same goals as in the corresponding main scenario. Alternative scenarios may also be used for error handling in cases in which the associated goals can still be fulfilled.
- ❑ *Exception scenarios:* Exception scenarios describe how the system must react in the case of a critical error during scenario execution that prevents fulfillment of the associated goal. Exception scenarios place particular emphasis on error recovery rather than on goal fulfillment.

Structuring scenarios

Additional information on scenario modeling can be found in [Pohl 2010] and [Potts 1995]. In the requirements viewpoint of the SPES modeling framework, SysML sequence diagrams [OMG 2010a] or ITU message sequence charts [ITU 2004] can be used to model this artifact type. During the requirements engineering process, it may be useful to model multiple scenarios. Scenarios can be structured using use cases ([OMG 2010a, OMG 2010bCockburn 2001], and use cases can be related to one another, for example, by means of include and extend relationships) or hMSCs [ITU 2004]. However, when using structuring scenarios in this way, the scenario specification must therefore document a complete behavioral specification.

Fig. 4-3 shows a SysML sequence diagram with a scenario model. The diagram depicts a scenario for executing a production process. This scenario fulfills one goal from Fig. 4-2. Furthermore, the model in Fig. 4-3 specifies five states that the SUD adopts during this interaction (for details, see Section 4.2.4).

Hint 4-3 lists the rules that have been defined in the requirements viewpoint for ensuring that the scenario artifacts have been modeled completely and correctly.

Rules for checking scenario models

Hint 4-3*: Rules for checking scenario models*

- ❑ Has a precondition been specified for each scenario?
- ❑ Does every scenario describe the entire interaction necessary to fulfill one or more goals?
- ❑ Does every scenario account for all actors that interact with the system?
- ❑ Have postconditions been specified for every scenario?

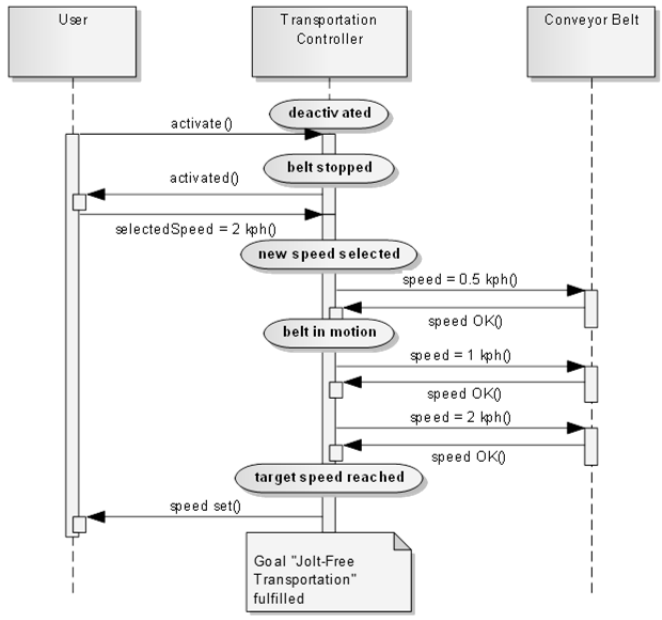


Fig. 4-3 *Example of a sequence diagram*

4.2.4 Solution-Oriented Requirements Model

Solution-oriented requirements are solution-specific descriptions of behavior, operations, and the information structure of the solution concept developed (see [Pohl 2010] and [Davis 1993]). They thus represent a first step towards the implementation. Solution-oriented requirements consist of a structural requirements model, an operational requirements model, and a behavioral requirements model. Solution-oriented requirements can thus be derived from scenario descriptions as scenarios may specify states that the SUD adopts after a certain interaction sequence has been executed. Furthermore, the operational requirements model and the structural requirements model of solution-oriented requirements can be derived in part based on scenarios and the context model, as both specify information that is exchanged between the SUD and the context and show how information is transformed from input to output.

Complementary development of the three perspectives

All three types of solution-oriented requirements models are developed complementarily as they present separate but interrelated aspects of the same SUD. A more detailed explanation of solution-oriented requirements is given in [Pohl 2010].

In the requirements viewpoint of the SPES modeling framework, SysML block definition diagrams can be used as static/structural models,

SysML activity diagrams can be used to model operational requirements models, and SysML state machine diagrams can be used to model behavioral requirements models. In the following sections, an example is given for each model type along with a brief explanation and the rules for checking each model type.

Structural Requirements Model

Purpose and example of structural requirements models

Fig. 4-4 shows an information structure model for an example system as a SysML block definition diagram. As shown, the static/structural requirements model gives a closer account of the information that is exchanged along the interfaces in the context model (see Section 4.2.1) and in part by the scenario model (see Section 4.2.3). Static/structural requirements models must therefore be defined consistently to both artifacts and can be used to document relationships between the objects pertaining to the information structure and other artifacts. For example, if a context model specifies the object "work piece data" to be exchanged between the SUD and its context, structural requirements models can be used to refine what information item "work piece data" consists of, e.g.: material type, length, width, height, and weight.

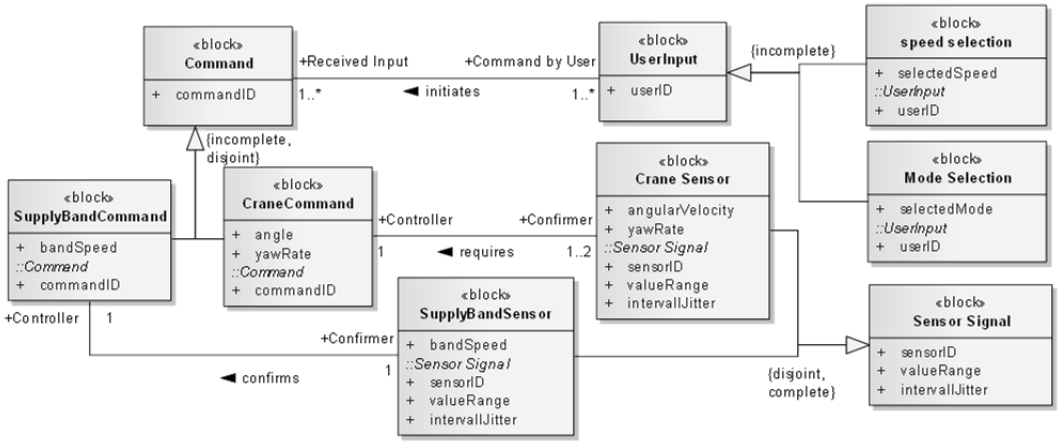


Fig. 4-4 *Example of an information structure diagram*

Hint 4-4 lists the rules that have been defined in the requirements viewpoint for ensuring that the structural requirements models have been modeled completely and correctly.

Hint 4-4: *Rules for checking structural requirements models*

- ❑ Have all inputs to the system from the context and its context entities (as specified in the context and scenario models) been accounted for?
- ❑ Have all outputs from the system to the system context and context entities (as specified in the context and scenario models) been accounted for?
- ❑ Have all information structures that are specified in behavioral and operational requirements models been documented?
- ❑ Have useful, non-trivial relationships (such as generalizations, aggregations, compositions) been introduced between information objects?

Rules for checking structural requirements models

Operational Requirements Model

Purpose and example of operational requirements models

Fig. 4-5 shows a SysML activity diagram as an example of an operational requirements model. This artifact type models operations that are derived by assigning user functions to the goals specified in the goal models (see Section 4.2.2) with reference to the interactions specified in the scenario models (see Section 4.2.3). Operational requirements models can therefore be seen as the solution-specific counterpart of the solution-neutral scenario artifacts. Consequently, the operations specified in the operational requirements models implement the functionalities that can be experienced by context entities (i.e., actors or external systems) through the interfaces that the system has with the context entities. As a result, the interfaces specified herein must be consistent to the interfaces specified in context models (see Section 4.2.1). This is similar to the functional black box model in the functional viewpoint (see Section 5), however, in contrast to the functional viewpoint, operational requirements models are partial requirements models that document the system's interaction with the context in more detail than scenario models. On the other hand, the functional viewpoint documents the entirety of the system's functions in order to foster analysis. As a consequence, artifacts specified in the functional viewpoint are based on the solution-oriented requirements models, particularly on the operational requirements models.

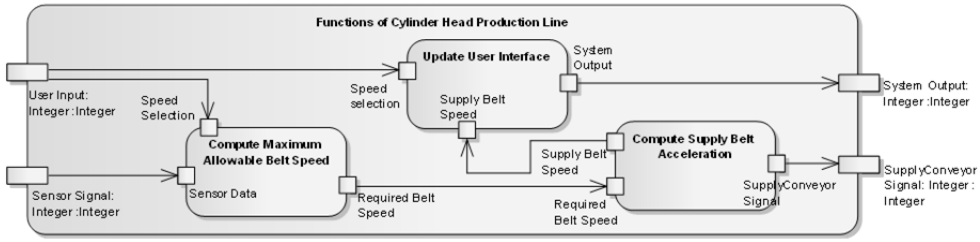


Fig. 4-5 *Example of an operational requirements diagram*

Hint 4-5 lists the rules that have been defined in the requirements viewpoint for ensuring that the operational requirements models have been modeled completely and correctly.

Rules for checking operational requirements models

Hint 4-5*: Rules for checking operational requirements models*

- ❑ Have all relevant system functionalities that have to be implemented by the SUD to fulfill its goals been considered?
- ❑ Have inputs and outputs been defined for every operation in the operational requirements models?
- ❑ Are the specified interfaces consistent to the interfaces in the context models?

Behavioral Requirements Model

Purpose and example of behavioral requirements models

Behavioral requirements models can be used to specify preconditions that must be in effect for system operations to be executed or postconditions that have to be fulfilled after an operation has been executed. Fig. 4-6 shows an example of a behavioral requirements model as a SysML state machine diagram. In this diagram, the states were partially derived from the scenario models (see Section 4.2.3). Transitions were also derived from the scenario models and completed during the specification of the requirements artifact at hand. Since the white box model of the functional viewpoint (see Chapter 5) uses state machines for specifying the internal behavior of functions, those behavioral models are based on the behavioral requirements models of the requirements viewpoint.

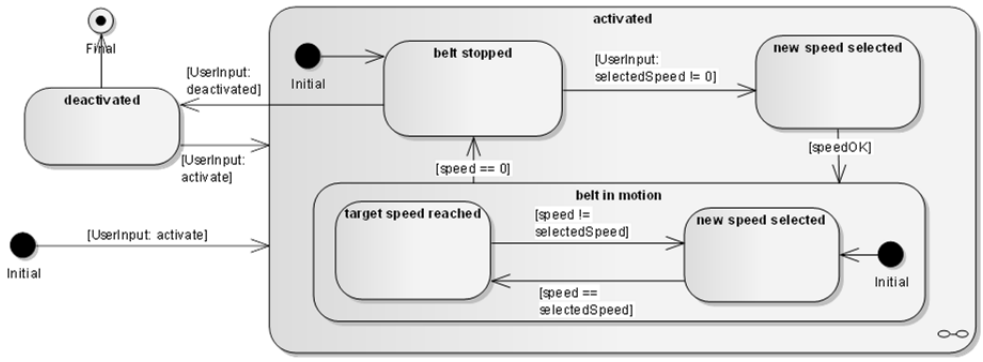


Fig. 4-6 *Example of a behavioral requirements diagram*

Hint 4-6 lists the rules that have been defined in the requirements viewpoint for ensuring that the behavioral requirements models have been modeled completely and correctly.

Rules for checking behavioral requirements models

Hint 4-6: *Rules for checking behavioral requirements models*

- ❑ Have all trigger events been considered in the behavioral requirements models?
- ❑ Have all system states and transitions of the SUD been considered?
- ❑ Do the behavioral requirements models specify preconditions and postconditions for scenarios?
- ❑ Do the behavioral requirements models specify activation conditions for operations?

4.3 Integration in the SPES Modeling Framework

This section gives an account of why some model types can be used in multiple viewpoints and how these model types have to be interpreted in the viewpoints (Section 4.3.1). Furthermore, this section explains how the requirements viewpoint can be integrated into the SPES modeling framework with regard to other viewpoints (Section 4.3.2) and different abstraction layers (Section 4.3.3). Hint 4-7 gives a short summary of correspondence rules that ensure consistency between the artifacts developed in each step.

Correspondence rules for the models in the requirements viewpoint

Hint 4-7*: Correspondence rules*

Context models ↔ scenario models

- ❑ Each actor included in any scenario has to be included in the context model
- ❑ Each actor included in the context model has to be included in at least one scenario model
- ❑ Inputs and outputs between the SUD and any actor have to be consistent in context and scenario models

Goal models ↔ scenario models

- ❑ Each scenario has to be related to at least one goal
- ❑ Each goal has to be fulfilled by at least one scenario

Goal models ↔ requirements models

- ❑ For each goal, system properties (functions, behavior, information structures) have to be defined in the requirements models related to the goal for fulfillment
- ❑ Each property documented in the requirements models must be related to at least one goal

Scenario models ↔ requirements models

- ❑ Each scenario must be capable of being processed based on the requirements models
- ❑ The inputs and outputs from the scenario models have to be consistent with the requirements models

Between requirements models

- ❑ The entry condition for each function defined in the function model has to be defined in the behavior model
- ❑ The information structure of the inputs and outputs of functions in each function model have to be defined in the information structure model
- ❑ The state-based actions and the transition-based actions belonging to the behavior model have to be described as functions in the function model

4.3.1 Use of Models across Viewpoints

Same model type, different viewpoints

The various model types used in the requirements viewpoint are also used in the other viewpoints. However, depending on the viewpoint, the model types have vastly different meanings and document entirely different information. For example, if a statechart were used in both the requirements viewpoint and the logical viewpoint, the statechart in the requirements viewpoint would represent the captured requirements and would summarize a possible solution with regard to the requirements. On the other hand, in the logical viewpoint, the statechart would represent a part of the logical architecture and it would detail how the system will be implemented rather than how it could be implemented. Similarly, functional models are used both in the functional viewpoint and in the requirements viewpoint. While in the requirements viewpoint operational

requirements models represent a type of partial functional model, the functional viewpoint is more concerned with an integrated, functional view on the entire SUD.

4.3.2 Integration across Viewpoints

Requirements viewpoint is the starting point for development

The requirements viewpoint is the starting point for the development process using the SPES modeling framework. Once requirements engineering activities in the requirements viewpoint have reached a satisfactory stability, the development process continues with the activities in the functional viewpoint (see Chapter 5). The functional viewpoint takes the scenario models and the solution-oriented artifacts from behavioral and functional requirements models as input and derives an approximate functional architecture that meets the requirements outlined in the artifacts. Further input from the requirements viewpoint is given to the logical and technical viewpoints (see Chapters 6 and 7 respectively). The logical viewpoint takes the system context and goal artifacts from the requirements viewpoint and derives a logical architecture that meets quality requirements defined in these requirements viewpoint artifacts. These artifacts also provide quality requirements for the technical viewpoint. In addition, the technical viewpoint suggests a concrete hardware/software architecture based on the requirements, functional, and logical viewpoints.

4.3.3 Integration across Abstraction Layers

All requirements artifacts on all abstraction layers

One key feature of the SPES modeling framework is the hierarchy of abstraction layers (see Section 3.4.1). Specifying requirements on different abstraction layers is a proven approach to reducing the complexity of development projects [Braun et al. 2010]. The requirements viewpoint therefore allows specification of all requirements artifacts. At each abstraction layer, the same set of artifacts is developed (i.e., context models, goal models, scenario models, and solution-oriented requirements models; see Section 4.2). The abstraction layers differ from one another with regard to the level of detail contained within their respective requirements artifacts, such that some abstraction layers contain more coarsely specified requirements (in the following, called higher abstraction layers) and some layers contain more detailed requirements (lower abstraction layers).

SUD decomposition in other viewpoints

The logical and/or technical viewpoints structurally decompose the SUD into subsystems. The decomposed subsystems that are structurally significant (e.g., important control units or safety-critical subsystems)

become the new focus of development and are hence treated as if they were the SUD on the next lower abstraction layer. The requirements process (see Section 4.4) starts anew for all of these subsystems.

4.4 The Requirements Process Model across Abstraction Layers

The following briefly illustrates an idealized development process that outlines the development of the different artifacts over time.

Step 1: Analyze and document the system context

- *1st Step: Analyze and document the system context:* Firstly, the system context in which the SUD will be used is analyzed and documented. The context of any subsystem consists of relevant parts of the context of the SUD as well as other subsystems of the SUD that the subsystem under development interacts with.

Step 2: Analyze and document goals

- *2nd Step: Analyze and document goals:* After modeling the system context, goals for the subsystem under development are elicited, documented, and negotiated with the stakeholders identified during context analysis. For the development of subsystems specifically, the documented goals must be consistent with those documented for the SUD. In detail, this means that the fulfillment of the goals of the SUD is dependent on the fulfillment of all goals of all of its subsystems.

Step 3: Define and model scenarios

- *3rd Step: Define and model the scenarios of system usage:* After the context and goal models have been sufficiently documented, scenarios are used to describe possible ways to fulfill the goals. The scenarios and goals have to be related: each goal has to be fulfilled by at least one scenario and each scenario must fulfill at least one goal. The development of goals and scenarios is a highly iterative and incremental process. Scenarios may lead to further goals not discovered in the first step. New goals will lead to further scenarios. This process continues until no new goals or scenarios are discovered. Scenarios of any subsystems depict refinements of scenarios of the SUD.

Step 4: Specify solution-oriented requirements

- *4th Step: Specify solution-oriented requirements:* Once the system scenarios are sufficiently documented, and each goal is fulfilled by one scenario, the solution-oriented requirements can be modeled. The system still considered as a black box. Use operational, structural, and behavioral requirements models to describe the SUD from the perspective of the context entities. Modeling should focus

on idealized system properties and essential interfaces of the system. Hence, the developed models should be neutral to specific implementation details, but should give closer accounts of *how* the aspects modeled in context, goal, and scenario models are achieved. The modeling of the SUD is a highly iterative and incremental process. It may be possible that, for example, new scenarios (i.e., scenarios missing from the second step) are identified during this step. These newly discovered scenarios may lead to new goals, and so on. This step terminates when no more changes are necessary in the artifacts. Quality requirements are documented relative to the appropriate solution-oriented requirements by means of appropriate annotations. In order to elicit these quality requirements, dedicated analysis steps may be necessary (see Chapter 9).

4.5 References

[Alfaro and Henziger 2001] L. de Alfaro, T. A. Henzinger: Interface automata. In: Proceedings of the 8th European Software Engineering Conference ESEC/FSE-9, 2001.

[Braun et al. 2010] P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Müller, B. Penzenstadler, K. Pohl, T. Weyer: Guiding requirements engineering for software-intensive embedded systems in the automotive industry. Computer Science - Research and Development. DOI: 10.1007/s00450-010-0136-y, 2010.

[Cockburn 2001] A. Cockburn: Writing Effective Use Cases. Addison-Wesley, 2001.

[Davis 1993] A. M. Davis: Software Requirements – Objects, Functions, States. 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Hammond et al. 2001] J. Hammond, R. Rawlings, A. Hall: Will it work? In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01), IEEE Computer Society Press, Los Alamitos, 2001, pp. 102-109.

[ITU 2004] International Telecommunication Union: ITU-T Z.120: Message Sequence Chart (MSC), 2004.

[Jarke and Pohl 1994] M. Jarke, K. Pohl: Requirements engineering in the year 2001 – (Virtually) managing a changing reality. Software Engineering Journal, Vol. 9, No. 6, 1994, pp. 257-266.

[Lamsweerde 2009] A. van Lamsweerde: Requirements Engineering – From System Goals to UML Models to Software Specifications. Wiley, West Sussex, 2009.

[Leveson 2000] N. Leveson: Intent specifications – An approach to building human-centered specifications. IEEE Transactions on Software Engineering, Vol. 26, No. 1, 2000, pp. 15-35.

[Lynch and Tuttle 1989] N. A. Lynch, M. R. Tuttle: An introduction to input/output automata. CWI Quarterly, Vol. 2, 1989, pp. 219-246.

[McMenamin and Palmer 1984] S. M. McMenamin, J. F. Palmer: Essential Systems Analysis. Prentice Hall, London, 1984.

[OMG 2010a] Object Management Group: OMG Systems Modeling Language™ (OMG SysML) Language Specification v1.2. OMG Document Number: formal/2010-06-02.

[OMG 2010b] Object Management Group: OMG Unified Modeling Language™ (OMG UML), Infrastructure v2.3. OMG Document Number: formal/2010-05-03.

[Pohl 2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, Germany, 2010.

[Potts 1995] C. Potts: Using schematic scenarios to understand user needs. In: Proceedings of the ACM Symposium on Designing Interactive Systems – Processes, Practices, Methods and Techniques (DIS'95). ACM, New York, 1995, pp. 247-266.

[Reisig 1991] W. Reisig: Petri nets and algebraic specifications. Theoretical Computer Science, Vol. 80, No 1, 1991, pp. 1-34.

[Weyer 2011] T. Weyer: Kohärenzprüfung von Anforderungsspezifikationen: Ein Ansatz zur Prüfung der Kohärenz von Verhaltensspezifikationen gegen Eigenschaften des operationellen Kontexts. Südwestdeutscher Verlag für Hochschulschriften, 2011.

[Yu 1997] E. Yu: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), IEEE Computer Society Press, Los Alamitos, 1997, pp. 226-235.

4.6 Acknowledgements

The authors would like to thank their former colleagues Dr. Kim Lauenroth (now with adesso AG) and Dr. Ernst Sikora (now with Automotive Safety Technologies GmbH) for their support in early phases of this research.

Andreas Vogelsang
Sebastian Eder
Dr. Martin Feilkas
Daniel Ratiu

5

Functional Viewpoint

The major concern of the functional viewpoint is to provide a formal and model-based behavior specification for the system under development. Therefore, the viewpoint provides two model types that structure the behavioral requirements according to user functions and provide an abstract realization of these. A user function captures a set of solution-oriented requirements, as specified in the models of the requirements viewpoint, and integrates them into a functional black box model — a behavioral description of the entire system under development. By using formally founded models, the functional black box model provides the basis for detecting undesired interactions between user functions at an early stage of the development process. User functions are later refined by a functional white box model that decomposes a user function into functions that represent smaller units of functionality and provide an abstract realization of the user function. Due to the high level of abstraction, this viewpoint is a step towards closing the gap between semiformal requirements and a formal system design.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_5,

5.1 Introduction

The functional viewpoint has two model types: functional black box model and functional white box model

The starting point for the functional viewpoint is a set of requirements for the behavior of the SUD provided by the models of the requirements viewpoint, especially the context model, the scenario models, and the behavioral requirements models (see Chapter 4). These models provide a complete set of requirements in a semiformal, model-based representation. The functional viewpoint provides two model types: the functional black box model, which formalizes the requirement models as user functions and integrates them into a comprehensive system specification [Broy 2010], and the functional white box model, which provides a decomposition of the user functions from the functional black box model into smaller functional units in order to give an abstract description of the realization of the user functions.

The models that are provided by the requirements viewpoint describe requirements for the SUD from the view of a specific usage context. The functional viewpoint translates these partial usage models into the notion of user functions that define the intended system behavior, including all interactions and dependencies between them. Thus, the result of the functional viewpoint is a comprehensive system specification.

A user function hierarchy consists of user functions and dependencies between them.

Within the functional black box model, the requirements models are translated into user function hierarchies consisting of user functions and dependencies between them (see Fig. 5-1 for an informal representation). Each user function realizes a piece of black box functionality and is defined by its syntactic interface and its behavioral specification. The syntactic interface comprises the ports via which the user function is connected to its context, and the behavioral specification defines the messages exchanged on these ports.

User functions may have quite complex functional requirements that may not even give any information about a possible solution for this requirement. In order to reduce this complexity and also to facilitate reuse of existing partial solutions, the user functions are refined in a functional white box model. This model consists of a set of functions that give an abstract solution of the functionality that is required by the user function.

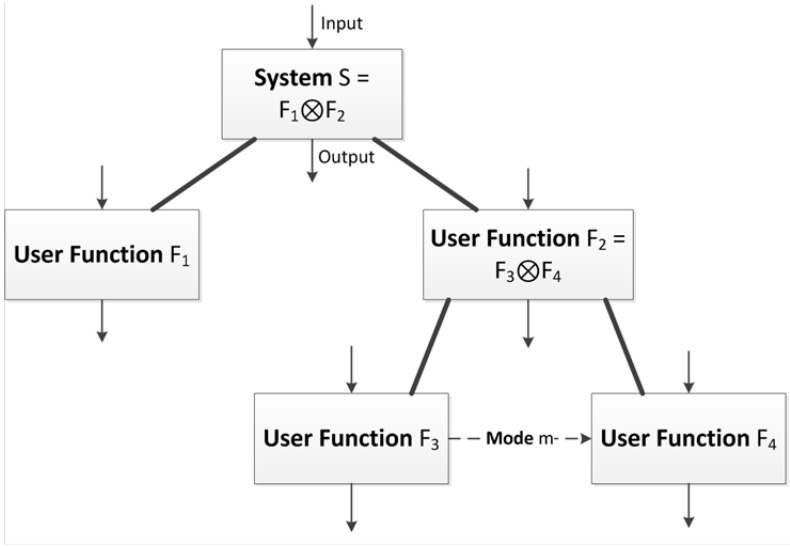


Fig. 5-1 *Informal representation of a user function hierarchy: user functions are composed into more complex user functions taking account of their dependencies (dashed horizontal arrows) and finally into the specification of the entire SUD.*

5.2 Concerns

Aims of the functional viewpoint

The central aims of the functional viewpoint are:

- Consolidating the functional requirements by formally specifying the requirements of the system behavior from the black box perspective
- Mastering feature interaction: detection and resolution of inconsistencies within the functional requirements
- Reducing complexity by structuring the functionality hierarchically from the user's point of view
- Understanding the functional interrelationships by collecting and analyzing the interactions between different (sub-) functionalities.

The functional viewpoint provides a hierarchically structured specification of the SUD behavior as it is perceived by the user at the system boundary (also known as usage behavior). In this context, a user may be a person but also another system. The functional viewpoint comprises the formal definition of the SUD interface with surrounding systems and users. The behavior of the entire SUD is then specified from the black box perspective by describing the exchange of messages between the SUD and its context. Here, the abstract data flow is specified, namely the intentional meaning of the exchanged data (as

opposed to the concrete message types). By formally describing the requirements, we create the basis for measuring the completeness of and detecting inconsistencies in the requirements, especially for interacting requirements of different functions (cf. feature interaction [Zave 1993]).

The functional viewpoint is independent from realization

The overall system functionality can be obtained from the composition of user functions (with respect to the dependencies between them). Here, the decomposition/structuring is not guided by architectural or technical aspects but is executed merely along the functional aspects required by the users.

Thus, an informal requirement can be realized by one or several user functions and a user function can realize one or more informal requirements. The structuring and refinement of the requirements models in the functional viewpoint makes it possible to analyze existing requirements and thus to detect and resolve inconsistencies (e.g., feature interaction) and missing requirements.

5.3 Functional Black Box Model

User functions

The central construct of the functional black box model is a user function that defines a part of the behavior of the SUD that can be observed at the system boundary. Fig. 5-2 shows how user functions are related to the black box interface of the SUD. Consequently, a user function does not contain any information about how it is implemented in the SUD.

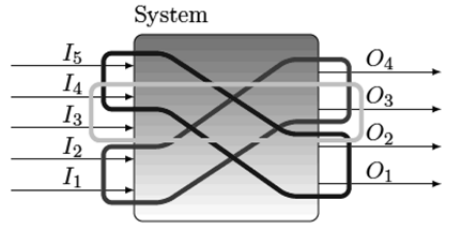


Fig. 5-2 *The system interface is structured into three user functions that all comprise a subset of input and output channels and their related behavior.*

Each user function has a syntactic interface that consists of a number of typed ports that are either input ports or output ports. In general, input and output ports define messages that an actor sends to the SUD or receives from the SUD. We also assign an interface behavior to each user function. A possible formalization of this notion can be found in [Broy 2010].

Example 5-1: Transportation user function

Fig. 5-3 and Fig. 5-4 illustrate a simple transportation belt user function. The user function has two input channels transmitting two values that indicate whether a work piece is present at the beginning or the end of the transportation belt. The output channel controls the transportation belt motor. The user function formalizes the requirement that the transportation belt shall be switched on if a work piece is present at the beginning of the belt but no other work piece is waiting at the end of the belt. The behavior specification is given by a simple table specification that maps input values to output values. The "?" character is an abbreviation for all possible values on this channel.

Fig. 5-3 *Syntactic interface of the Transportation user function*

IndicatorPos1	IndicatorPos2	BeltMotor
WPPresent	NoWP	Activate
?	WPPresent	Deactivate

Fig. 5-4 *Interface behavior of the Transportation user function described by a simple I/O table that maps values of the input channels to values of the output channel*

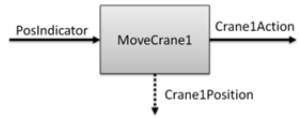
Mode channels

Fig. 5-2 also reveals that user functions can have common input or output channels. This indicates a certain kind of dependency between user functions. This dependency expresses that a user function also needs information about another function's state or input values to determine the correct output values. In order to model dependencies between user functions, we use special ports that connect user functions via a *Mode Channel*. These channels are called mode channels as they usually transmit values that represent an abstract state of the SUD — a mode.

In principle, we could avoid the appearance of functional dependencies completely by explicitly all inputs ever relevant for a user function and its output behavior to syntactic interfaces. However, due to the high number of dependencies in a system, this leads to a completely confusing and unmanageable behavior. Therefore, it is better to capture the dependencies using modes.

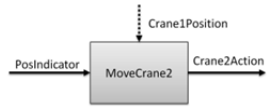
Example 5-2: Dependent crane user functions

An automation system has two cranes that transport work pieces through an assembly line with six stations: **S**upply **B**elt, **Mi**lling, **Gr**inding, **Me**asuring, **As**sembling, and **D**elivery **B**elt. Two user functions steer the cranes. However, to avoid collisions, Crane 2 is not allowed to approach station "Measuring" when Crane 1 is approaching that station. Thus, the user function "MoveCrane2" depends on the state of the user function "MoveCrane1." We model this dependency by introducing a mode channel "Crane1Position" and extend the behavior specification of the user function "MoveCrane2" to prevent the user function approaching the "Measuring" station if Crane 1 is approaching it. The resulting user functions and their behavior specifications are given by Fig. 5-5 and Fig. 5-6.



PosIndicator	Crane1Action	Crane1Position
WP@SB ∧ ¬WP@Mi ∧ ¬WP@Gr	SB2Mi	Mi
WP@Mi ∧ ¬WP@Gr	Mi2Gr	Gr
WP@Gr ∧ ¬WP@Me	Gr2Me	Me

Fig. 5-5 *Syntactic interface and behavior specification of the user function "MoveCrane1." The current position of the crane is propagated by the mode channel "Crane1Position."*



Crane1Position	PosIndicator	Crane2Action
¬Me	WP@Me ∧ ¬WP@As	Me2As
?	WP@As ∧ ¬WP@DB	As2DB

Fig. 5-6 *Syntactic interface and behavior specification of the user function "MoveCrane2." Whether or not the crane can approach the "Measuring" station depends on the "Crane1Position" mode.*

User function hierarchy

We distinguish two kinds of user functions: *atomic* user functions and *composite* user functions. Composite user functions are composed of at least two subuser functions. User functions that are not decomposed are called atomic user functions. Atomic user functions must provide a behavior specification that defines their behavior observed at the system boundary. In a nutshell, atomic and composite user functions allow the developers to decompose the functional requirements, formalized as user functions, into a hierarchy of user functions.

Example 5-3: User function hierarchy of the Automation system

We can integrate the given user functions from Example 5-1 and Example 5-2 into a comprehensive system that provides all the user functions. The two user functions "MoveCrane1" and "MoveCrane2" are composed to a composite user function "MoveCranes." Fig. 5-7 visualizes this as a user function hierarchy. The syntactic interface and the interface behavior of the composite system are derived from the composition of the user functions. Fig. 5-8 illustrates the resulting syntactic interface of the automation system.

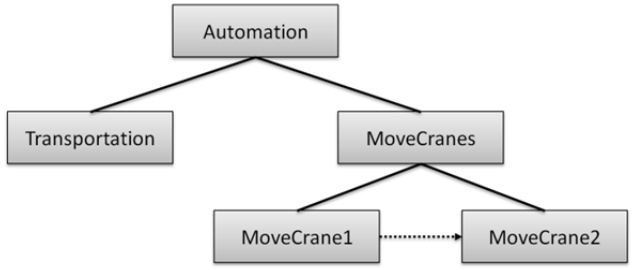


Fig. 5-7 *User function hierarchy of the composite Automation system. The dashed arrow represents the mode channel between the user functions "MoveCrane1" and "MoveCrane2." The solid arrows indicate a subfunction relationship.*

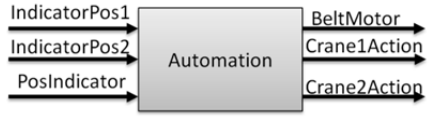


Fig. 5-8 *Syntactic interface of the composite Automation system, derived by the composition of the interfaces of the subfunctions. The mode channel between the user functions "MoveCrane1" and "MoveCrane2" becomes invisible when composing the user functions.*

All of the presented are taken from a comprehensive case study that has been modeled in SPES 2020 [Eder et al. 2011].

5.4 Functional White Box Model

The second model type within the functional viewpoint is the functional white box model. The purpose of this model is to provide a decomposition of the user functions from the functional black box model into smaller functional units in order to give an abstract description of the realization of the user functions. These smaller functional units are called functions.

Function

A function itself can be described by a syntactic interface with an associated interface behavior. This is an example of how we take advantage of a common underlying modeling theory in SPES that provides modeling concepts that are reused in each of the viewpoint models. The purpose of this decomposition is to master the complexity that arises from the black box specification, or, more specifically, the question of how the specified outputs will be computed from the given inputs.

Example 5-4: Functional white box model for a user function

In order to realize the behavior of the user function "MoveCrane2" as specified in Fig. 5-6, we have to decompose the user function into smaller functional units called functions. Fig. 5-9 shows the functional white box model for this decomposition. The user function is broken down into five functions. In contrast to the black box specification of the user function, the white box model already provides some information about the realization, e.g., the sensor values are read in separately and the control logic is encapsulated in a single function.

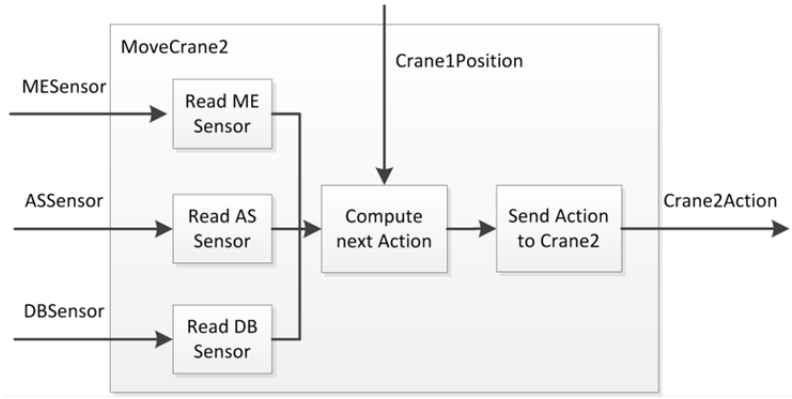


Fig. 5-9 *Functional white box model for the user function "MoveCrane2"*

The relation between the functional black box model and the functional white box model is as follows: the functions of the functional white box model together must show the same behavior as specified by the user function of the functional black box model. Therefore, it is necessary to provide a mapping between the inputs and outputs of the user function and the inputs and outputs of the functional white box model. This mapping can then be used to check whether the functional white box model conforms to the functional black box model.

Example 5-5: Mapping between black box and white box models

The user function "MoveCrane2" from Fig. 5-6 has only one input for indicating the position of work pieces and this is called "PosIndicator." The white box model as depicted in Fig. 5-9, however, has three inputs for indicating the position of work pieces: MESensor, ASSensor, and DBSensor. Therefore, in order to check whether the functional white box model specifies the same behavior as the functional black box model, mapping that maps the values of the inputs to each other is required. Fig. 5-10 gives this mapping in a tabular representation.

PosIndicator	MESensor	ASSensor	DBSensor
WP@Me ∧ ¬WP@As	WP	¬WP	?
WP@As ∧ ¬WP@DB	?	WP	¬WP

Fig. 5-10 *Tabular representation of the mapping between the inputs of user function "MoveCrane2" and its corresponding white box model*

Another important aspect of this white box model is reuse. The functionality that is captured by a function can be reused in several functional white box models. In our example, the "Read AS Sensor" function could be reused to decompose the user function "MoveCrane1." In such scenarios it is particularly important to check the conformity of the composed white box model to the original specification of the user function because reuse bears the risk of incorrectly reusing certain functions.

5.5 Analyses

The functional viewpoint offers a model of the functionality of an SUD at a very early stage of development. Several analyses and evaluations can be performed on this model to verify and validate the functional requirements of the SUD. Validation of requirements in this sense means that the requirements are reasonable with respect to each other, i.e., no contradictions between requirements, no unintended behavior due to unintended interactions between requirements, or no missing requirements. In contrast, verification ensures that the behavior as described in the model of the logical viewpoint fulfills the behavior that is specified in the model of the functional viewpoint.

In the following we outline some of the analyses that can be used for validation or verification:

Simulative validation

If the behavior is specified by executable models (e.g., state machines), the models of the functional viewpoint provide a functional prototype of the SUD. This prototype can be used to run through certain actions and scenarios to validate the behavior of the SUD in cooperation with the user. Thus, unintended behavior can be revealed and evaluated.

Detection of inconsistencies

The functional black box model of the functional viewpoint also allows the detection of inconsistencies. Inconsistencies are input patterns that cause different functions to output conflicting values. This can be checked prior to execution and thus reveal contradicting requirements [Harhurin 2010].

Completeness analyses

At the beginning of development, requirements only define part of the entire functionality. During development, more and more requirements complete the system specification. The models of the functional viewpoint facilitate this process by showing input patterns for which no output is defined. These situations represent insufficient specification that should be discussed and possibly refined.

Test cases

The functional viewpoint models and formalizes the functional requirements and thus serves as a specification. In this role, the model of the functional viewpoint can be used as a testing oracle. Test cases can be generated from it to verify that a model of the logical viewpoint fulfills the specified behavior of the functional viewpoint.

Tracing between user functions and logical components

The model of the logical viewpoint describes the inner structure of the system that implements the user functions. Input and output ports of user functions can be linked to logical components that implement these user functions. This yields a tracing relation between user functions and logical components, allowing design faults to be detected and functions to be tested individually, and ensuring the implementation of all user functions [Vogelsang et al. 2012].

5.6 Integration in the SPES Modeling Framework

Within the SPES modeling framework, the functional viewpoint is located between the requirements and the logical viewpoints. This section gives an overview of the relation and the differences between the functional viewpoint and its neighbors in the SPES modeling framework.

Relation to the requirements viewpoint

The goal of the requirements viewpoint is to capture the requirements for the SUD and to document their relation to the context of the SUD. The result is a set of models that represent a variety of requirements for the SUD. These models are derived from abstract goals and information about the system's context. A crucial task is now to integrate all of these

requirements models into a system specification that is consistent and sound. This task is done in the functional viewpoint (at least for the requirements that refer to system behavior). The requirements models of the requirements viewpoint are captured and refined as user functions that the SUD offers to its context. The complex interactions that arise from the interplay of the requirements (feature interaction) are explicitly modeled in the functional black box model of the functional viewpoint. The result is an integration of the isolated requirements models from the requirements viewpoint into a comprehensive system model that describes the required system behavior at its boundary to its context and serves as a specification. Technically, this relation is realized by reusing and refining the models of the requirements viewpoint. The context model that is defined in the requirements viewpoint defines the syntactic interface of the SUD, i.e., the inputs from the environment as well the outputs of the SUD to the environment. This interface is authoritative for the definition of user functions. The syntactic interface of a user function must always be a subset of the syntactic interface of the SUD as defined in the context model. Moreover, a behavioral requirements model of the requirements viewpoint can initially be reused as behavior of a user function. When integrating all user functions, the behavioral models must be enriched with additional behavior that arises from the concurrent integration in the SUD. The scenario models of the requirements viewpoint serve as test cases and validation conditions for the functional black box model of the functional viewpoint.

Relation to the logical viewpoint

The model of the logical viewpoint describes the internal logical structure of the SUD by means of communicating logical components. Thus, the viewpoint considers the SUD in a glass box view in contrast to the functional viewpoint where the SUD is considered as a black box. The relation between a user function (from the functional black box model) and a logical component (from the logical viewpoint) is often confused. A user function formalizes a part of the requirements that the SUD has to fulfill at its boundary, whereas a logical component captures a part of functionality that lies within the SUD and that contributes (amongst other things) to the realization of a user function. In general, there is an n:m mapping between user functions and logical components. This means that one user function can be realized by a number of logical components, and one logical component can contribute to the realization of a number of user functions. As this gap is sometimes confusing and hard to manage, the functional viewpoint provides the functional white box model. In this model the user functions from the functional black box model are decomposed into functions that give an abstract solution for the realization of the user function. The functions are then mapped to

a logical component of the logical viewpoint. In this way, a logical component contains a set of functions that contribute to the realization of one or several user functions.

Functional viewpoint and the logical Viewpoint across abstraction layers

Depending on what is considered the system under development on a certain abstraction layer, the functional black box model of the functional viewpoint provides a functional black box specification of the respective SUD. This model structures the interface specification of the system according to user functions of the system. As a consequence, we get new functional black box models for the functional viewpoint if we change the abstraction layer. In the uppermost abstraction layer, for example, the system is considered as a whole with one functional black box model that specifies the black box behavior of the system at its boundary. If we step into the next lower abstraction layer by decomposing the system into a number of logical components, each logical component can again be considered as a system (with a different context) and thus has an own functional black box model in the functional viewpoint. These models again specify the intended functional black box behavior for each logical component. An important perception here is to recognize that the user functions of the system in one abstraction layer do not have to be the same as the user functions of a logical component in the next lower abstraction layer. An example for this was already given in Section 3.5.2. In fact, the user functions of a logical component are heavily influenced by the functions of the functional white box model. When mapping the functions of the functional white box model to logical components, we determine user functions that the logical component must fulfill and that are part of the functional black box model of the logical component.

5.7 The Functional Viewpoint Process

In the following, we will give an idealized process through the functional viewpoint. Note that in reality, this process is highly iterative and also interweaved with the processes of the other viewpoints. It is also important to note that the requirements viewpoint process does not have to be completed before the functional viewpoint process is started, nor does the functional viewpoint process have to be completed before the process of the logical or technical viewpoint is started. Furthermore, we can divide the functional viewpoint process into building the functional black box model and building the functional white box model. We will start with the black box model that is subsequently refined in the white box model.

Step 1: Extract user functions

In the first step, we use the solution-oriented requirement models from the requirements viewpoint to extract user functions for the SUD. Initially we can translate each behavioral model of the requirements viewpoint into one user function. Later we might find it useful to merge a set of behavioral models into just one user function. We make sure that the inputs and the outputs of the user functions conform to the syntactic interface as defined by the context model of the requirements viewpoint, i.e., the inputs and outputs of a user function are a subset of the inputs and outputs of the context model. Defining user functions is not a canonical step. There are a variety of possibilities for structuring a system according to user functions. However, a guiding principle is to define the different functions as they are perceived by the user (similar to the notion of a use case).

Step 2: Structure user functions in a user function hierarchy

Once we have a set of user functions we try to structure them in a user function hierarchy. We may also introduce new user functions that group a set of user functions as their subfunctions. There are multiple ways to arrange the user functions into a user function hierarchy. The user function hierarchy should again reflect a structure of the user functions as they are perceived by the user. Another goal of choosing user functions and arranging them in a user function hierarchy is to gain a manageable set of user functions that, on the one hand are not too complex to give their behavior as a behavior specification, but on the other hand do not have too many dependencies to other user functions.

Step 3: Specify interfaces for all atomic user functions

The next step is to specify an interface for each atomic user function in the user function hierarchy. This is done by providing both a syntactic interface by means of input and output channels of the user function and a behavior specification. Behavior specifications can be given by any specification technique that an interface behavior abstraction can be assigned to. Examples for such specifications are state machines [Broy 2010], I/O tables [Thyssen and Hummel 2011], or data flow diagrams [Leuxner et al. 2010].

Step 4: Model dependencies using mode channels

The next step is to model dependencies and resolve inconsistencies between the user functions. Inconsistencies between user functions can have many facets. For example, two user functions that share a common output channel are in a conflicting situation as they both write values to that output channel at the same time [Harhurin 2010]. Dependencies on the other hand can be intended or unintended. Intended dependencies represent desired interaction between user functions such as one user function interrupts another or works differently depending on results of another user function. Unintended dependencies arise from unconscious interplay between user functions. Inconsistencies and dependencies are modeled in the functional viewpoint by means of mode channels. Thus,

we extend the user functions' interface with additional mode channels that transmit information necessary for resolving inconsistencies and modeling dependencies.

Step 5: Extend the behavior specifications

The last step of the functional black box model is to extend the behavior specifications of the user functions with respect to the mode channels introduced. This means that we have to integrate the information that is provided by the mode channels into the behavior of the user function. For two conflicting user functions, for example, a mode channel between them resolves this conflict by transmitting the information that one user function is currently not allowed to send a value over the common output channel.

Step 6: Building the functional white box model

From here, we start building the functional white box model. We do not have to wait for the black box model to be fully specified before commencing with the white box model. We could also start building parts of the white box model immediately after Step 1. We build a functional white box model for each atomic user function, and this model provides a high-level description of tasks that need to be performed in order to realize the user function and the data flow between these tasks. We capture such tasks as functions with a syntactic interface that defines the data that is processed and produced by the function and we additionally associate an interface behavior that specifies the behavior of the function.

Step7: Check conformance between white box and black box models

Finally, we need to ensure that the functional white box model conforms to the functional black box model and that this method is a valid realization for the user function. Therefore, we have to check that the composition of the functions in the functional white box model yields the same behavior that the user function from the functional black box model demands. Several methods, all with advantages and disadvantages, can be used for this purpose, for example, testing, model checking, or formal verification.

5.8 References

[Broy 2010] M. Broy: Multifunctional software systems: Structured modelling and dpecification of functional requirements. In: Science of Computer Programming, Vol. 75, No. 12, 2010, pp. 1193-1214.

[Eder et al. 2011] S. Eder, A. Vogelsang, M. Feilkas: Seamless modeling of an automation example using the SPES methodology. In: Technical Report TUM-I1110. Technische Universität München, May 2011.

[Harhurin 2010] A. Harhurin: From Interaction Patterns to Consistent Specifications of Reactive Systems. PhD Thesis, Technische Universität München, 2010.

[Leuxner et al. 2010] C. Leuxner, W. Sitou, B. Spanfelner: A formal model for work flows. In: SEFM 2010: Proceedings of the 8th International Conference on Software Engineering and Formal Methods, 13-18 Sept. 2010, pp. 135-144.

[Thyssen and Hummel 2011] J. Thyssen, B. Hummel: Behavioral specification of reactive systems using stream-based i/o tables. Software and Systems Modeling, DOI: 10.1007/s10270-011-0204-1.

[Vogelsang et al. 2012] A.Vogelsang, S.Teuchert, J.-F. Girard. Extend and characteristics of dependencies between vehicle functions in automotive software systems. Proceedings of the 2012 International Workshop on Models in Software Engineering, 2012

[Zave 1993] P. Zave: Feature interactions and formal specifications in telecommunications. In: IEEE Computer, Vol. 26, No. 8, 1993, pp. 20-28.

Sebastian Eder
Jakob Mund
Andreas Vogelsang

6

Logical Viewpoint

This section provides an outline of the logical viewpoint. This viewpoint describes the internal logical structure and the behavior of the system under development (SUD). The main task in the logical viewpoint is the distribution of functions to a hierarchy of logical components. The main model type of the logical viewpoint is the logical component architecture: it describes the logical components of the system and their relationship. The structure of this logical component architecture is often influenced by nonfunctional criteria, e.g., maintainability or reliability. In contrast to the technical viewpoint, the logical viewpoint does not focus on the technical infrastructure provided, e.g., the controllers or communication devices used.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_6,

6.1 Introduction

The logical viewpoint describes the logical structure and the distribution of responsibilities functionality of a system by means of a network of interacting logical components that are responsible for a set of functions. These logical components and their interactions are arranged in the logical component architecture of the system. The design of the logical component architecture is driven by various considerations, such as achieving maximum reuse of already existent components or fulfilling different nonfunctional properties of the SUD. The logical component architecture bridges the gap between functional requirements and the technical implementation. All examples in this chapter are taken from a case study modeled in SPES 2020 [Eder et al. 2011].

6.2 Concerns

Aims of the logical viewpoint

The main aims of the logical viewpoint are:

- Describing the internal logical structure of the SUD by partitioning the system into communicating logical components
- Allocating desired functions to cohesive logical units
- Supporting the reuse of already existent logical components and designing the logical components such that future reuse is facilitated
- Defining the total behavior of the system (as opposed to the partial behavior specifications in the models of the functional viewpoint) and enabling the complete simulation of the entire system

The logical components should be designed to capture the central domain abstractions and to support reuse. As a consequence, the logical component architecture should be as insensitive as possible to changes in the desired user functionality or technical platform. It should be the artifact in the development process with the highest stability and with the highest potential for reuse.

The functional black box model and the logical component architecture are two orthogonal structures of the system functionality. A brief comparison of both models is illustrated in Tab. 6-1.

Comparison of functional and logical viewpoints

In contrast to the functional black box model, the logical component architecture does not emphasize the formalization of the functionality that can be observed at the system boundary, but rather the structuring and partitioning of the SUD into communicating logical components. The behavior of these logical components as a whole realizes the

behavior determined by the functional viewpoint. The connection between these two models is established by the functional white box model of the functional viewpoint. A logical component represents a unit that provides one or more functions of the functional white box model. In other words: the functions of the functional white box model are distributed to logical components.

In the logical viewpoint, structuring is performed according to diverse criteria, such as the organizational structure within the company, nonfunctional requirements, or even according to the user function hierarchy of the functional black box model. However, it is important to note that the logical viewpoint abstracts from hardware details. Therefore, some (nonfunctional) requirements are better addressed in the technical viewpoint. Hint 6-1 summarizes the partitioning in the logical viewpoint.

Partitioning in the logical viewpoint

Hint 6-1*: Partitioning in the logical viewpoint*

- A core activity in the logical viewpoint is the partitioning of functions into communicating logical components
- Partitioning can be performed according to a user function hierarchy of the functional viewpoint, the organizational structure, or nonfunctional requirements
- Hardware details are part of the technical viewpoint rather than the logical viewpoint

The logical viewpoint provides a complete description of the system functionality, without, however, anticipating technical decisions with regard to implementation (e.g., the platform on which the logical components will be deployed).

Tab. 6-1 *Brief comparison of the functional black box model and the logical component architecture*

Functional Black Box Model	Logical Component Architecture
Problem domain	Solution domain
Black-box view of the SUD	White-box view of the SUD
Structured by user functions	Structured by architectural entities
Used primarily to specify what the SUD should do	Used primarily to design the SUD
Functional specification may overlap and must be checked for inconsistencies (horizontal decomposition)	Network of communicating logical components (vertical decomposition). Their composition must be checked against the desired system functionality.
Captures the functionality of the SUD	Works as a first cut at design
(Possibly) partial behavioral specification	Total behavioral specification

6.3 Logical Component Architecture

Logical components

The logical component architecture is the main model of the logical viewpoint and consists of logical components. These logical components are decomposed into other logical components, which results in an acyclic hierarchical structure of logical components, i.e., a tree. The leaves of this tree are called atomic logical components and they are not decomposed further.

Similarly to the models of the functional viewpoint, every logical component carries a syntactic interface consisting of typed ports that are either used for output or input. Input ports can be connected to output ports by channels and thus, logical components are linked together via channels.

Ports and channels

Logical components define these channels to connect their logical subcomponents. Every atomic logical component has to specify its (total) behavior. The behavior of an atomic logical component is defined directly (for example, using an automaton). The behavior of non-atomic components is derived from the composition of their subcomponents.

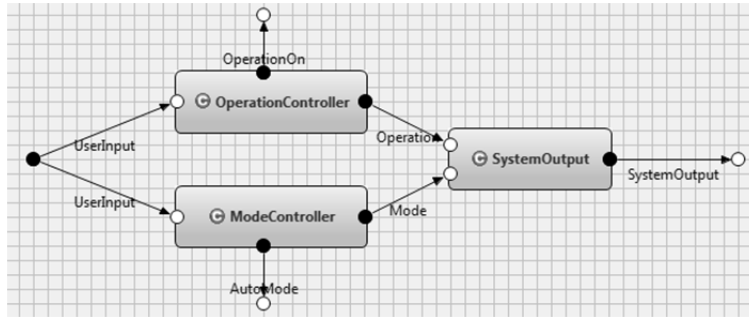


Fig. 6-1 *Example of a logical component architecture*

Example 6-1: User input controller

Fig. 6-1 shows an example of a logical component architecture that is part of the model of the logical viewpoint. The architecture models a user input controller consisting of three logical components: OperationController, ModeController, and SystemOutput. The output ports are illustrated as black circles placed at the borders of the logical components. White circles depict input ports. The ports not drawn at the border of any logical component represent input or output ports to the next higher logical component in the hierarchy. Output ports are connected to input ports via channels, as depicted by the arrows. The labels next to the channels associate a name with the corresponding channel. Note that each channel has a specific type, but this is not shown in Fig. 6-1.

Hint 6-2*: Using the metamodel of the logical viewpoint*

- Entities for the distribution of functions are logical components that are the basic building blocks representing the behavior.
- Each logical component offers a syntactic interface that consists of (typed) ports. Ports can furthermore be input or output ports for a given logical component.
- Communication between logical components is achieved by connecting output ports to input ports by means of channels for which the types of the ports must coincide.

Usage guidelines for the logical viewpoint

6.4 Analyses

The logical viewpoint provides a model of the system design independent of any hardware decision. However, this model is expressive enough to allow a variety of analyses to be performed on it. Ultimately, the logical viewpoint aims at constructing models that are so close to the final implementation that the code can be generated completely from the models of the logical viewpoint. Further analyses that can be performed on the models of the logical viewpoint are:

- *Complete simulation:* Since the model of the logical viewpoint defines a total system behavior, it allows comprehensive simulation of the SUD. While a simulation of the functionality is already possible in the functional viewpoint, the logical viewpoint also allows several nonfunctional properties to be checked (e.g., reliability and abstract timing constraints) by simulating the SUD.
- *Verification of system properties:* Properties that have been specified previously (e.g., in the functional viewpoint) can be verified automatically using model checking techniques.
- *Test case generation:* As the model defines the total behavior of the SUD, this enables test cases to be generated, for example, by following [Pretschner et al. 2004].
- *Concurrency analyses:* For a couple of years, there has been an ongoing paradigm shift from single-core towards multicore processors. Employing multicore architectures for embedded systems brings several advantages but also poses new challenges for software engineering. The logical viewpoint allows analysis of the logical component architecture in order to determine an adequate parallel hardware platform and an adequate deployment of logical components to hardware components. Thereby, adequacy can refer to several system requirements (e.g., response time, robustness, or hardware costs).

6.5 Integration in the SPES Modeling Framework

Within the SPES modeling framework, the logical viewpoint resides between the functional and the technical viewpoints. This section gives an overview of the relation and the differences between the logical viewpoint and its adjacent viewpoints.

6.5.1 Functional Viewpoint

Relation to the functional viewpoint

The functional white box model of the functional viewpoint describes an abstract solution of the system. The goal is to realize the desired functionality as specified in the functional black box model using a set of abstract functions. The logical components of the logical component architecture comprise a set of these abstract functions in order to structure them. Therefore, the functions of the functional white box model must be distributed to the logical components of the logical viewpoint. In addition to this direct relation, it is also possible to relate the logical components to the user functions of the functional black box model by assessing the logical components that are involved in the realization of a user function. This relation can be used for impact analyses and other validation methods (cf. [Vogelsang et al. 2012]). Further details on the correlation between a user function and a logical component can also be found in Section 5.6.

6.5.2 Technical Viewpoint

Relation to the technical viewpoint

The technical viewpoint describes the hardware topology of the system using technical components such as control units and busses. Logical components defined in the model of the logical viewpoint are deployed on the technical components described by the model of the technical viewpoint, and communication channels between logical components are realized either internally on one controller or by busses or other technical communication facilities. Logical components are typically deployed as atomic units and are thus rarely split up into tasks that can then be seen as atomic units that are deployed. However, the modeling theory does not forbid this and this results in an n:m relation between logical components and technical components.

6.5.3 Abstraction Layers

Logical viewpoint and the abstraction layers

The relation of the logical viewpoint model over the abstraction layers can be described as follows: a change of the abstraction layer in the logical viewpoint corresponds to a decomposition of a logical component or the entire system into logical subcomponents. In this case, the logical viewpoint model serves as a structural characteristic for the lower abstraction layer, which means that, on the lower abstraction layer, viewpoint models exist for each logical subcomponent. The input and output data that is processed by the logical viewpoint model of a certain abstraction layer must also appear in the logical viewpoint model of the next lower abstraction layer; in fact they must be processed by the logical subcomponents. In more complex development scenarios, for example, when suppliers are involved, the change of an abstraction layer determined by one decomposition step of the system might not be handled that strictly. Instead, the transition to the next abstraction layer can, for example, be interpreted as handing over a certain part of the system to a supplier or another department within the company. In this scenario, several decomposition steps will be performed in one abstraction layer until the parts are handed over to a supplier or another department whose models reside in the next lower abstraction layer. However, the aforementioned relation between these abstraction layers also holds in this scenario. The supplier must provide viewpoint models for the specific part that is in his responsibility.

6.6 The Logical Viewpoint Process

Step 1: Adopt system boundary from the functional black box model

To develop the logical viewpoint of a system, the system boundary from the functional black box model is adopted and can be refined, if necessary. The system is the root logical component of the decomposition tree containing all logical components. The system boundary is expressed by a syntactic interface.

Step 2: Decomposition into logical subcomponents

The next step, which is one of the most crucial steps in developing the model of the logical viewpoint, is the decomposition of the SUD into logical components. The result of the decomposition is a tree of logical components, with the complete SUD as its root. This decomposition is oriented on nonfunctional requirements and quality aspects, and not solely on the functional requirements of the SUD. Thus, a logical component is not a user function, but a logical unit that is required to implement the system's functionality.

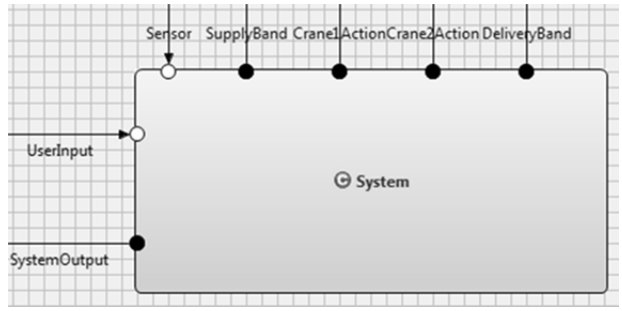


Fig. 6-2 *The system is the root logical component of the logical component hierarchy.*

Step 3: Connect components via channels

The different logical components of a system can be linked together via communication channels. These channels should only link logical components that really are intended to communicate. To enable logical components to be connected, every logical component is enriched by input and output ports that form the syntactic interface of the logical components. The ports are then used to connect logical components via channels.

> **Example 6-2**: Decomposition of the SUD
>
> Fig. 6-3 shows the decomposition of the SUD from Fig. 6-2. The system is decomposed into three logical components: IOAdapter, Transportation-Controller, and UserInteraction. The decomposition is guided by separation of concerns, separating the IO, the transportation unit and the user interface. The ports at the system boundary in Fig. 6-2 correspond to those ports in the decomposed network in Fig. 6-3 that do not belong to any of the logical components.

Step 4: Specify behavior for all leaf components

The logical components that are not decomposed any further (leaves in the logical component tree) are then enriched with behavior. This can be done using several techniques: table specifications [Thyssen and Hummel 2011], state machines [Broy and Stølen 2001], data flow diagrams [Leuxner et al. 2010], or mathematical functions, to name just a few. Unlike the models of the functional viewpoint, logical components define total behavior. This means that a logical component has to have behavior specified for every input. The composition of the behavior of all leaves results in the behavior of the complete system.

Therefore, the SUD can be simulated and the perceived behavior can then be compared to the specification of the system and the models of the functional viewpoint. If necessary, the logical component architecture can be refined or altered by running through the steps above.

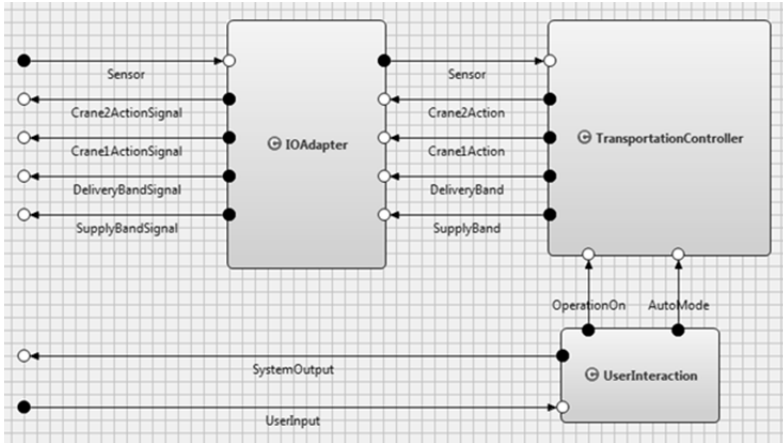


Fig. 6-3 *Decomposition of the SUD into a logical component architecture*

6.7 References

[Broy and Stølen 2001] M. Broy, K. Stølen: Specification and development of interactive systems: focus on streams, interfaces, and refinement. Springer, 2001.

[Eder et al. 2011] S. Eder, A. Vogelsang, M. Feilkas: Seamless modeling of an automation example using the SPES methodology. In: Technical Report TUM-I1110. Technische Universität München, May 2011.

[Leuxner et al. 2010] C. Leuxner, W. Sitou, B. Spanfelner: A formal model for work flows. In: SEFM 2010: Proceedings of the 8th International Conference on Software Engineering and Formal Methods, 2010.

[Pretschner et al. 2004] A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel: Model-based testing for real. In: International Journal on Software Tools for Technology Transfer, Vol. 5, No. 2, 2004.

[Vogelsang et al. 2012] A.Vogelsang, S.Teuchert, J.-F. Girard. Extend and characteristics of dependencies between vehicle functions in automotive software systems. Proceedings of the 2012 International Workshop on Models in Software Engineering, 2012

[Thyssen and Hummel 2011] J. Thyssen, B. Hummel: Behavioral specification of reactive systems using stream-based i/o tables. Software and Systems Modeling, 2011.

Raphael Weber
Philipp Reinkemeier
Dr. Stefan Henkler
Dr. Ingo Stierand

7

Technical Viewpoint

In the technical viewpoint, the system under development (SUD) is modeled in terms of resources: some of the resources are software resources (such as a scheduling slot); others are hardware resources (such as a computing resource) that are associated with a hardware description (e.g., a processor). In this viewpoint, hardware and software are explicitly separated. Thus, tasks have to be modeled along with their processing resources (the resource on which they are executed). Tasks themselves are scheduled by a scheduling resource that must be connected with the task it schedules.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_7,

7.1 Introduction

From platform-independent to platform-specific

The technical viewpoint is mostly concerned with the question of how to get from the platform-independent models (logical components) of the logical viewpoint to platform-specific models (technical components).

The strict separation of application development and deployment to a distributed technical architecture enables reuse of standardized models and thereby selection of commercial off-the-shelf platforms for hosting the different application parts. This clean separation is an underlying principle in many other approaches as well [Kleppe et al. 2003, Mellor et al. 2004, Giese and Henkler 2006, AUTOSAR VFB 2010].

In the technical viewpoint, the properties of the selected technical components and their resources provided are modeled. Furthermore, modeling in this viewpoint is intended to address the following concerns: Which logical components of the application are realized by which resources? How do these components share (time and space) the resources? This sharing of resources by applications is a key to reducing the cost of the developed product.

Interference between applications: real-time analysis, common cause analysis

However, the sharing of resources also introduces interference between logical components that were independent in the logical viewpoint and this interference must be analyzed and checked against requirements of the SUD. Real-time analysis reveals whether real-time requirements (e.g., end-to-end latency) imposed in the logical viewpoint can be met by the selected technical architecture and the chosen deployment. A common cause analysis, particularly a common mode analysis, is performed to detect failures that affect independent paths in the logical architecture.

More generally, the following concerns are addressed in the technical viewpoint:

- Which (composition of) resources (e.g., computation, communication, IO devices such as sensors and actuators, etc.) are present in the system, how are they connected, and which properties (e.g., HW properties such as memory size or clock frequency) do they have?
- What resource-consuming and resource-offering entities are present in the system (e.g., software tasks consume computation and communication resources)?
- Which consuming resources are deployed on which offering resources, which mapping is required (type, behavior), and which design constraints are given?

- What is the specific platform-specific behavior and interface of a logical component?
- What are the specific scheduling requirements (e.g., task 1 needs input every 20ms, which platform configuration can fulfill all timing constraints: interrupt, latency, end-to-end, responses, and so on)?

7.2 Metamodel of the Technical Viewpoint

Model elements used in the technical viewpoint

In this section, we introduce the fundamental concepts of the technical viewpoint metamodel. However, it is not within the scope of this document to give a complete specification of the metamodel. A comprehensive specification is given in [Weber et al. 2012, Baumgart et al. 2011].

In the technical viewpoint, the electric/electronic architecture of the system is specified, and this comprises mechanical and hydraulic components controlled by a network of control units that the logical components shall be allocated to.

The electronic control units may be software-programmable hardware components or application-specific integrated circuits (ASICs). Mechanical components are, for example, cams, shafts, switches, and relays. Hydraulic components include, for example, valves and cylinders.

The concepts provided for describing a view corresponding to the technical viewpoint are more specialized than those provided for modeling a requirements viewpoint, a functional viewpoint, or a logical viewpoint. This is because the intention is to apply dedicated analysis techniques for checking satisfaction of contracts based on models of technical artifacts [Damm et al. 2011].

The metamodel provides concepts for specifying the technical viewpoint of a system by means of technical components. While we do not provide specialized concepts in the metamodel for mechanical or hydraulic components, their aspects however can still be specified by means of contracts [Baumgart et al. 2011]. This allows their dynamics to be characterized, and also allows the assessment of whether requirements of functions are fulfilled by the mechanical or hydraulic components and the electronic components controlling them.

In SPES 2020, the focus is on software-intensive systems. Therefore, the metamodel incorporates specialized concepts for describing the resources of a network of electronic control units that enables execution platforms hosting the components of the logical viewpoint to be modeled. Here, resource limitations come into play, e.g., the sharing of computing resources by multiple logical components. The behavior of

the allocated logical components will therefore depend on the properties of the resource as well as the other logical components allocated to the very same resource.

7.2.1 Resources

Resources: consuming and offering

The concepts of the metamodel for specifying the technical viewpoint of a system are inspired by the UML Profile for MARTE (Modeling and Analysis of Real-Time embedded systems) [Object Management Group 2009].

The concept of a resource constitutes an abstraction of the resources a platform provides. Additionally, resources represent the allocated behavior from logical components. Examples of a resource include:

- Computational resources, i.e., the computational power of a processor
- Bandwidth of a communication resource
- Storage capacity of a storage resource

As some of these resources might be shared by multiple consumers, resource-sharing can be explicated by means of schedulers. A scheduler distributes fractions of a resource according to a given scheduling strategy. The concept of a scheduler allows both static and dynamic scheduler types to be modeled.

Hint 7-1*: Check rules for the resource model*

- What kind of resources can a system platform provide? Why would it make sense to connect two computational resources by means of a communication resource?

7.2.2 Schedulers

How exactly resources are shared: schedulers

A scheduler distributes fractions of a resource according to a given strategy. The clients of a scheduler are modeled by the concept of scheduler slots. The resource, whose capacity is shared and assigned by a scheduler, is typically expressed using a processing resource. A scheduling policy allows specification of the strategy of the scheduler, e.g., fixed-priority. The scheduler slots, which receive a fraction of the resource, are typically aggregated to the scheduler parameter specification that determines parameters used by a scheduler to run its strategy. Such parameters are, for example, the definition of a priority or the maximum size of a time slice.

The concept that actually uses the fraction of the processing resource is a concurrency resource or specializations thereof. If a schedule was determined a priori, it can be directly specified as the schedule by means of an expression. This overrides whatever was specified in the scheduling policy.

The protocol between the sharing controller and resource consumers

The relation of the concepts scheduler, scheduler slot, and concurrency resource is modeled by special scheduler ports. These ports must be typed by a port specification that conforms to the scheduler port specification concept. This specification comprises five flows that are referenced in the following roles:

- *Activate event:* An incoming event flow that denotes the request for the activation of a concurrency resource.
- *Start event:* An outgoing event flow that denotes the granting of a scheduler on a previously received activate event.
- *Finish event:* An incoming event flow that denotes the release of the capacity of the processing resource by the concurrency resource. It is triggered when the concurrency resource has finished its computation or communication.
- *Suspend event:* An outgoing event flow that shall be triggered by a scheduler or forwarded by a scheduler slot when access to the processing capacities of a scheduled resource has been granted before, but shall now be revoked in favor of another scheduler slot.
- *Resume event:* An outgoing event flow that shall be triggered by a scheduler or forwarded by a scheduler slot when access to the processing capacities of a scheduled resource has been granted and suspended before, and now access to the processing resource is granted again.

The scheduling of a processing resource may look as depicted in Fig. 7-1. The ports with dotted edges are scheduler ports. The figure also shows how to use the scheduler and the scheduler slot concepts when modeling the hierarchical scheduling of a processing resource. A scheduler slot can also serve a second scheduler that then only distributes the capacity of the processing resource that it receives through its underlying slot in the main scheduler to its clients. An example of a scheduler can also be seen in Fig. 7-3.

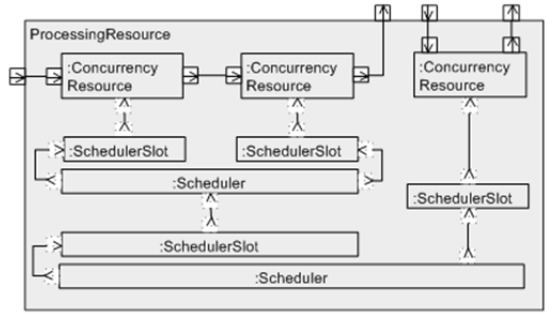


Fig. 7-1 *Example of using the scheduler concept*

Hint 7-2*: Check rules for the scheduler model*

- Do the scheduling parameters of each scheduler slot match the type of policy of the scheduler that it is connected to? For example: Does a scheduler using a fixed-priority strategy require the definition of a priority per slot?

7.2.3 Computing Resources

Processing power offering computation resources

The concept of a computing resource represents a processing device capable of storing and executing tasks. It is an abstraction of elements of an execution platform hosting multiple tasks that require computing capacity in order to perform their computations. Being derived from a processing resource, a computing resource may be subject to scheduling of its computing capacity.

The task concept models a computation task. Since it is derived from the concurrency resource concept, it can be connected to a scheduler slot. Thus, it uses the computing capacity the scheduler slot receives from a scheduler. A task can aggregate multiple specifications of its execution time that have been measured or analyzed previously. Provided that a task would have exclusive access to a computing resource, the execution time is the time from start to finish of the task. As the execution time may vary depending on the type of computing device, multiple execution times can be specified, each with a unique identifier. An example specification of an execution time would be: {ARM7; 500µs; 1ms}, where 500µs is the best base execution time and 1ms is the worst case execution time. An example of a computing and a communication resource is depicted at the bottom of Fig. 7-3.

7.2.4 Communication Resources

Data transporters offering communication resources

The concept of a communication resource represents an abstraction of a communication device of an execution platform capable of transferring information from one location to another. The attributes of a communication resource are used to describe its most relevant properties needed for timing analysis of a technical architecture. Being derived from the concept of a processing resource, a communication resource may be subject to scheduling of its communication capacity.

The concept of a frame denotes a data item that is transmitted by a communication resource. As the concept is derived from the concurrency resource concept, it can be connected to a scheduler slot. Thus it may use the communication capacity the scheduler slot receives from its scheduler. An example of a communication resource is given in Fig. 7-3.

7.2.5 Data Encapsulation

Layers of signals, messages, and frames

We consider the technical architecture as a network of computing resources that are interconnected by means of communication resources. However, the frames that are transmitted by these communication resources are not the data items that would be sent directly by an application task. In most execution platforms, an application programming interface (API) that hides the complexity of different bus technologies and communication protocols is provided. Data sent by application tasks is then processed by a communication stack that is typically organized in layers as in the Open Systems Interconnection model (OSI).

Aside from the intention to abstract from the details of a layer of communication protocols from the logical viewpoint, in the technical viewpoint, the real-time aspects or safety aspects of the logical viewpoint have to be specified as well. This allows a conclusion to be drawn about whether requirements regarding the communication of components of the logical viewpoint can be fulfilled when they are allocated to a technical architecture. Fig. 7-2 shows an example of the concepts that allow the data encapsulations performed by a communication stack to be captured. For the sake of brevity, the scheduler slots and a scheduler are not shown in this figure.

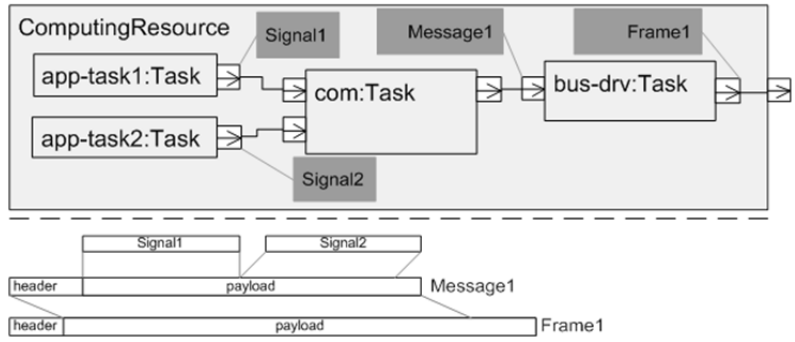


Fig. 7-2 *Example for specifying the data encapsulation*

The signal concept represents data sent by application tasks to which components from the logical viewpoint have been allocated. The declared signal references ports in the technical architecture model to indicate ports at which the signal is sent or received. The concepts message and frame also allow referencing of multiple ports. Data encapsulation by a communication stack as illustrated in the bottom half of Fig. 7-2 is expressed by the concept of mapping between messages and signals and frames and messages. The mapping describes the composition of messages by frames and of signals by messages. The condition that requires a task that is responsible for the encapsulation of signals in messages to actually trigger the message can be defined by the attributes of signal-to-message mapping. For example, some signals mapped to a message may cause it to be triggered, while others just update the value inside the message.

7.2.6 Tasks

The primitive computational unit: a task

A task is a specialization of the concurrency resource concept denoting a computation task. The computing capacity needed by the task is provided either directly by a computing resource (see Section 7.2.3) or by a scheduler slot (see Section 7.2.2).

Hint 7-3: *Check rules for the task model*

- ❑ Is every task connected to a scheduler slot by means of scheduler ports with reversed directions (task requires a scheduler port specification — the slot provides it)?
- ❑ Does each task have at least one execution time specification?
- ❑ Does each task have an execution time specification for the type of processor that matches the type of the computing resource on which it is executed?

7.3 Mapping between Viewpoints and Abstraction Layers

Relation between logical and technical viewpoints

The SPES modeling framework introduces mapping links that allow component realizations between abstraction layers and component allocations between viewpoints to be traced.

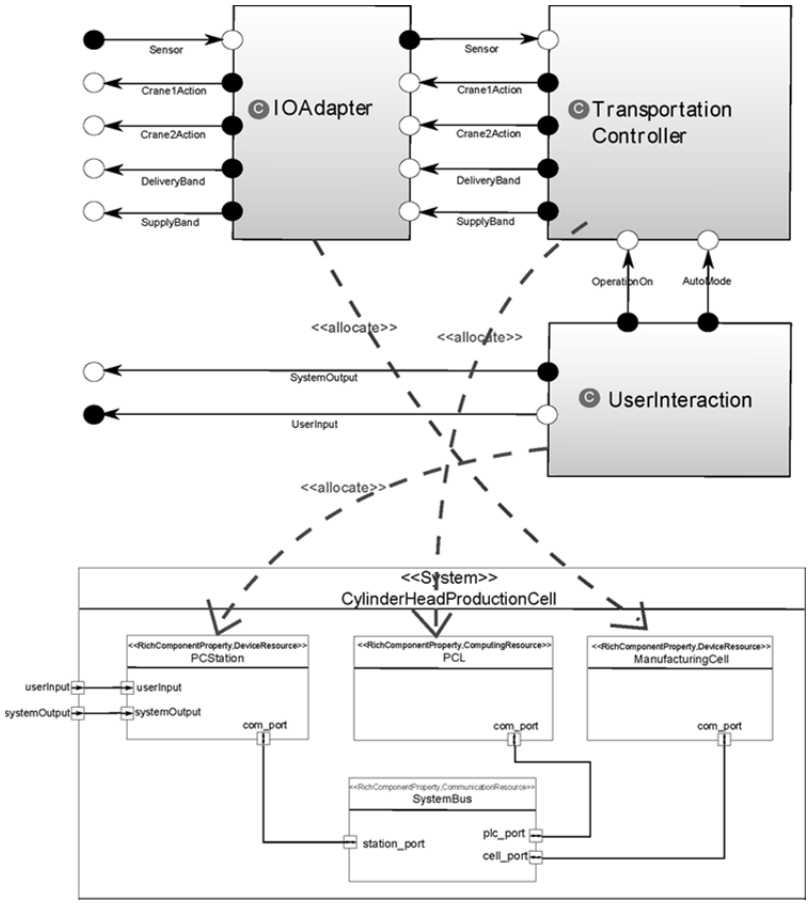


Fig. 7-3 *Example of a mapping between logical and technical viewpoints*

A mapping specifies correlations of observable behavior of components (e.g., logical and technical) by formal specification of how the dynamics of ports (interaction points) of one component are projected onto corresponding behavior of ports of another component. A component can also be represented by another (modeling) element that specifies the behavior of interactions. In order to formally check the validity of a mapping, a mapping block is defined. This mapping block specifies how

the behaviors of the mapped parts relate to each other (i.e., how the behavior of component *C1* is represented as the behavior of component *C2*, for each mapping component *C1* and *C2*). For more details on how mappings are represented in the SPES modeling framework, we refer to [Weber et al. 2012].

Fig. 7-3 shows an example of a mapping between the logical (top) and technical viewpoints (bottom). Here, the mapping links are visualized by the dotted arrows between components of the logical and the technical viewpoints. For the sake of simplicity, the mapping block is not shown.

Since the mapping relates component parts, the context of both these parts has to be described. Therefore, the concept of instance references, which has been inspired by the AUTOSAR and EAST-ADL metamodels, is used.

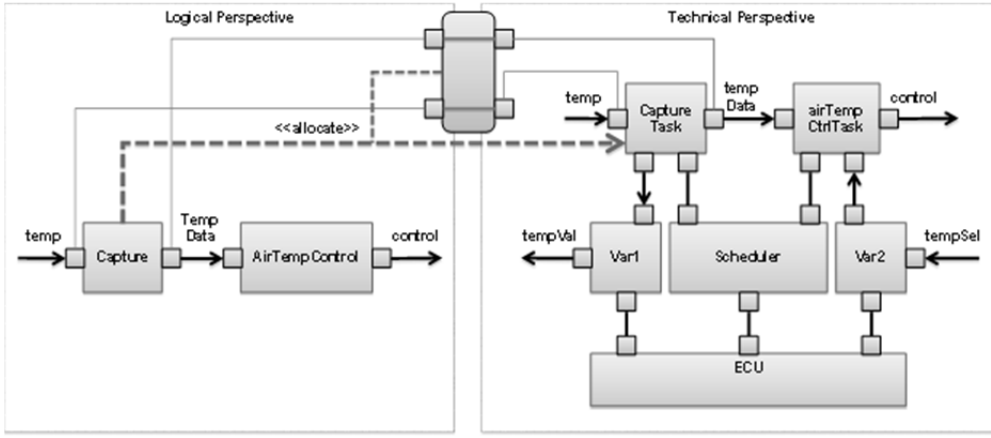


Fig. 7-4 *Example mapping between the logical and technical viewpoints*

Fig. 7-4 shows an example of a mapping between the logical and technical viewpoints (an allocation). The ports of the *Capture* component are mapped to the ports of the *Capture* task in the technical viewpoint. The mapping block is represented as a gray box between both viewpoints. As the links to the mapping block indicate, this is a simple direct mapping. This enables us to check completeness and receptiveness.

Completeness is defined as follows: Given a set of contracts C linked to components in the logical viewpoint, assuming flows/services of the ports of those components are allocated to flows/services of resources, the mapping must involve all flows/services mentioned in any of the contracts in C.

Receptiveness is defined as follows: An allocation of a model of the logical viewpoint to a model of the technical viewpoint shall not restrict the behavior of the model of the technical viewpoint. Generally, mappings can be more complex, as shown, for example, in [Gezgin et al. 2010].

7.4 How to Get from the Logical to the Technical Viewpoint

Step-by-step method to model in the technical viewpoint

In this section we will describe the steps necessary to start the model of a technical viewpoint from a finished logical viewpoint model. Once both models are finalized they have to be mapped to each other.

- We start by defining the hardware architecture as it may already exist or as we plan to construct it. The hardware architecture thereby does not need every detail present in the real world (it is a model after all). For the model, we use resources to represent hardware units.
- Next, we differentiate between computing resources (such as processors) and communication resources (such as buses). Our model elements in this model layer are used to separate actual computation units and data transfer units since they usually use different sharing approaches.
- In this step we model tasks, scheduler slots, and scheduler(s) in each computing resource and connect tasks among each other or to external elements. Communication resources are enriched by frame triggers (similar to tasks), and again scheduler slots and a scheduler.
- Now we connect all scheduler slots to their schedulers for each communication and computation resource. Consequently, we have to define the scheduling policy or the schedule in each scheduler. Additionally, depending on the actual scheduling policy, further information such as priorities has to be defined in each scheduler slot. Now each frame trigger and tasks are deployed to their respective scheduler slots by connecting their scheduler ports.
- The final step includes defining how signals, messages, and frames are composed and whether a signal or a message triggers a frame or not. In computing resources, we need to specify the execution times for each task and which task of a task port is mapped to which signal in a communication resource.

Once these five steps have been completed, the mapping from the logical to the technical viewpoint has to be defined. In addition, refinements of

the contracts of upper abstraction layers or the logical viewpoint in the current level may be annotated to tasks or resources. Here, timing requirements concerning end-to-end latencies, deadlines, or specific activation behaviors are most appropriate since potential timing analysis properties were only introduced in the technical viewpoint.

7.5 References

[AUTOSAR VFB 2010] AUTOSAR GbR. Specification of the virtual functional bus, Version 2.1.0, October 2010.

[Baumgart et al. 2011] A. Baumgart, E. Böde, M. Büker, W. Damm, G. Ehmen, T. Gezgin, S. Henkler, H. Hungar, B. Josko, M. Oertel, T. Peikenkamp, P. Reinkemeier, I. Stierand, R. Weber: Architecture modeling. In: OFFIS Technical Report, OFFIS Oldenburg, March 2011.

[Damm et al. 2011] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, I. Stierand: Using contract-based component specifications for virtual integration testing and architecture design. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) 2011, 14-18 March 2011, pp. 1-6.

[Gezgin et al. 2010] T. Gezgin, R. Weber, M. Girod: A refinement checking technique for contract-based architecture designs. In Proceedings of the 4th International Workshop on Model-Based Architecting and Construction of Embedded Systems (ACES-MB 2011) at the 14th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2011), 2011.

[Giese and Henkler 2006] H. Giese, S. Henkler: A survey of approaches for the visual model-driven development of next generation software-intensive systems. In: Journal of Visual Languages and Computing, Vol. 17, No. 6, pp. 528-550.

[Kleppe et al. 2003] A. Kleppe, J. Warmer, W. Bast: MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional, Boston, 2003.

[Mellor et al. 2004] S. J. Mellor, S. Kendall, A. Uhl, D. Weise: MDA Distilled. Principles of Model-Driven Architecture. Addison-Wesley Professional, Redwood City, 2004.

[Object Management Group 2009] Object Management Group. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Version 1.0, November 2009.

[Weber et al. 2012] R. Weber, E. Thaden, P. Reinkemeier, A. Baumgart. Specification of an architecture meta-model. In: OFFIS Technical Report, OFFIS Oldenburg, January 2012.

Kai Höfig
Dr. Mario Trapp
Bastian Zimmer
Prof. Dr. Peter Liggesmeyer

8

Modeling Quality Aspects: Safety

Safety is a central quality property of embedded systems. While progress in development methodologies, techniques, and tools enable the developer to manage the rapidly growing system complexity, this has long not been true for safety engineering methodologies. A promising approach to advancing the state of the art in safety engineering for software-intensive embedded systems lies in the application of model-driven development concepts to traditional safety engineering approaches. This chapter gives an overview of how safety analysis models can be integrated seamlessly into design artifacts and how model-driven development concepts can enable the modular safety assurance of platforms

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_8,

8.1 Introduction

Safety is typically defined as freedom from unacceptable risk (of harm). To ensure a certain level of quality, in most industrial domains the development of safety-critical systems is governed by standards. As mirrored in those standards, the development of a safety-critical system affects almost all process steps in a development lifecycle, ranging, for example, from requirements engineering through functional aspects to the technical design. For this reason, safety is not represented in a single viewpoint but as a quality aspect in the SPES modeling framework that has a crosscutting influence and is integrated into several viewpoints.

8.2 Concerns

Tight integration between safety analysis models and system development models

The growing complexity of safety-critical embedded systems is leading to increased complexity in safety analysis models. It is therefore not appropriate to develop functionality and consider safety in separate tasks. Safety aspects have to be integrated as tightly as possible into the development process and its models. Since model-based development of embedded systems deals with the increased complexity of such systems, their safety analysis has to follow this approach as well. Therefore, the goals of the quality aspect safety are:

- Provide modular, hierarchical, and model-based safety analysis models to "keep pace" with a state-of-the-art model-based development
- Ensure consistency and traceability among safety analyses in multiple views and on different layers of abstraction
- Ensure consistency between model-based safety analyses and other model-based development artifacts

The solution provided by this quality aspect is divided into two parts. The first part is component-integrated component fault trees (in short, C²FTs), as introduced in Section 8.3. Component fault trees (CFTs) provide the benefits of a modular and hierarchical safety analysis, whereas the component integration extension allows for a tight coupling between a component fault tree and a logical component. Based on C²FTs, this quality aspect contains two additional methods, one for managing consistency and protecting intellectual property over several layers of abstraction, as presented in Section 8.3.1, and one for

calculating probabilistic worst case execution times (pWCET), as presented in Section 8.3.2.

The second part of the quality aspect supports the system developer with the safety-related deployment of logical components onto technical components and is presented in Section 8.4. Deploying a logical software component onto a computer platform requires a check of whether the computer platform provides the safety requirements demanded by the software component.

8.3 Component-Integrated Component Fault Trees

The benefits of a hierarchical decomposition of complex systems and the applied principle of *separation of concerns* using (logical as well as technical) components can be perfectly transferred to the safety analysis model using the approach of component-based abstraction in fault tree analysis presented here. This decreases the complexity of safety analysis models, increases the connection between safety and development, and thereby reduces development costs.

The hierarchical decomposition of the system under development (SUD) into components and subcomponents, as well as the communication among them, follows a metamodel as specified by the logical viewpoint (see Chapter 6) but can also be applied to other model-based development languages. The central model element is the component that is embedded into a hierarchy of subcomponents and supercomponents communicating with each other using ports and connections.

Each component has an associated safety view and the propagation of failures follows *port interconnections*

The integrated metamodel for component fault trees follows this generic component model and allows the relation of a separate component fault tree for each component of the system's hierarchical decomposition. Fig. 8-1 shows the metamodel for component fault trees to be integrated with the component model: here, a component fault tree element, labeled CFT, is related to the component element of the metamodel. Faults propagate from one component to another via their interconnections (modeled using ports and connections). This is also reflected within the component fault tree metamodel: input and output failure modes are related to port elements of the component model [Domis and Trapp 2009].

The method supports the developer with automated modeling features

Based on these models, automations support the developer in handling the various component fault tree model elements such as gates, input and output failure modes, and edges between them to model the

failure behavior of the system under development using Boolean logic.

The approach was evaluated in the course of the SPES 2020 evaluation studies (see Chapter 16). Evaluation results show a good acceptance in this industrial area, since component fault trees are a model-based extension of widely accepted fault trees. Furthermore, measurements taken show that the complexity of modeling component fault trees compared to modeling classic fault trees is decreased, and the readability of component fault tree model elements is increased compared to classic fault trees. The evaluation was done by experiment in industry with more than ten people, from both the system development and safety analysis fields, and also in academia with the same amount of people and a similar background.

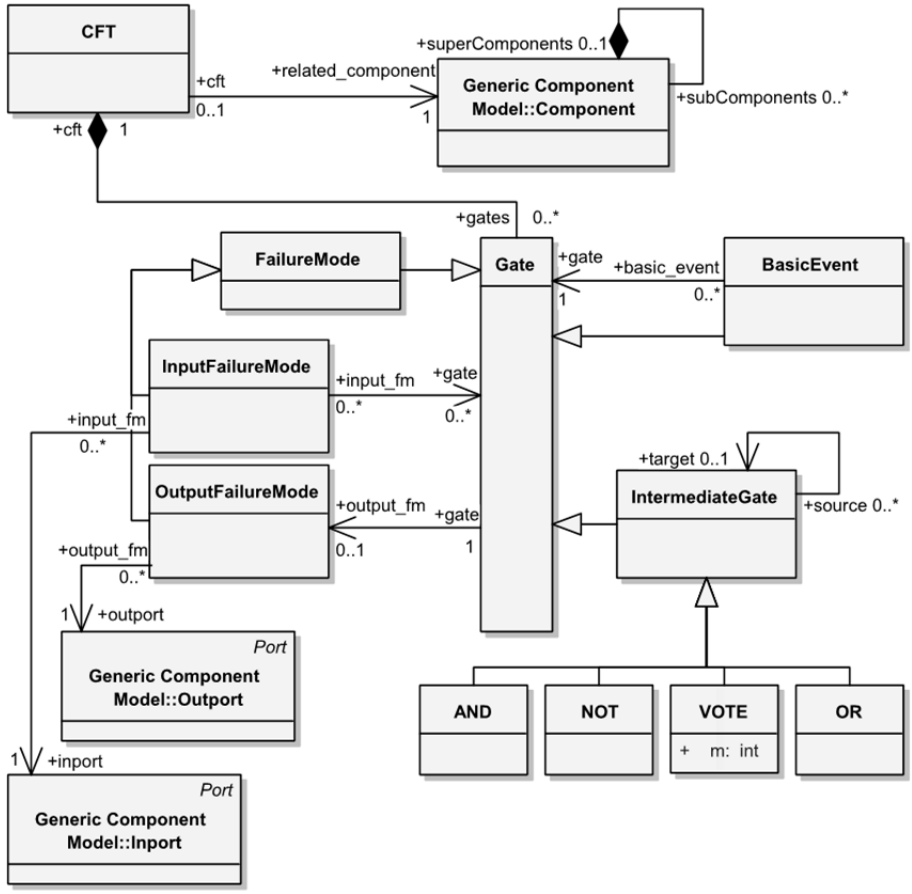


Fig. 8-1 *Metamodel for component fault trees*

8.3.1 Protecting IP and Managing Consistency across Abstraction Layers

Since the development process of modern embedded systems is often spread over many different stakeholders, e.g., different companies, this aspect is a special issue in safety analysis. During the hierarchical decomposition, components have a *specification* given by one stakeholder and are subsequently implemented by a different stakeholder using logical and technical subcomponents that *realize* the functionality as specified for the supercomponent. To analyze the failure logic of such a component, the failure logic of the supercomponent is also realized by the component fault trees of its subcomponents. To protect the intellectual property of the implementing stakeholder from the specifying stakeholder, the *white box* safety view of the subcomponents' fault trees can be transformed into a protective *black box* safety view using Boolean reduction [Domis et al. 2010]. In this way, the stakeholder of the supercomponent can still achieve the failure propagation of the specified component, but the intellectual property contained in the implementation is protected. Furthermore, the methodology allows checks regarding whether realization and specification are consistent with each other in order to ensure the consistency of the modular and hierarchical safety analysis across multiple abstraction layers.

8.3.2 Failure-Dependent Timing Analysis

This section presents a methodology that takes advantage of the tight integration of safety analysis models and system development models to combine elements of both worlds for a new execution time analysis approach.

The integration of component fault trees makes safety information accessible to other safety-related analyses such as worst case execution time analysis

Embedded real-time systems are growing in complexity and resource demand that today goes far beyond systems with simplistic closed-loop functionality. Current approaches of worst case execution time (WCET) analysis are used to verify deadlines of such systems, but these approaches calculate or measure the WCET as a single value that is used as an upper limit for a system's execution time. Overestimations are taken into account to make this upper limit a safe limit, but modern processor architectures expand these overestimations into unrealistic dimensions.

Here, therefore, probabilities of safety analysis models are combined with elements of system development models to calculate a *probabilistic* worst case execution time (pWCET). Safety analysis models are used in this approach as a source for probabilities [Adler et al. 2010]. Since safety analysis models typically reflect the occurrence of failures and

their propagation through the system under development, our approach aims at mechanisms in systems that are executed in *addition* to a failure. Such mechanisms usually belong to the area of fault tolerance and detect or process an error [Höfig et al. 2010, Höfig 2011b]. In this way, very unlikely time-intensive execution scenarios can be identified. This type of system becomes certifiable for a lower execution time if a deadline has to be guaranteed for a certain probability.

Example 8-1: Table lookup

An example of such a system is depicted in Fig. 8-2. This example is a reduced version of an example from the automotive domain as presented in [Höfig 2011a]. The right-hand side of Fig. 8-2 depicts the architecture of the system using logical components, and the left-hand side shows a part of the component fault tree model. The system measures sensor data with *Sensor A*. If this data is within a given range (plausibility test), the measured data is taken as the output of the system. If the test judges the data to be erroneous, data from a different sensor is taken to estimate the data for *sensor A* using a table lookup. Therefore, the execution of the table lookup function depends on whether *Sensor A* produced erroneous data or whether there is a failure in the *Test* component. Knowing the probability of occurrence for the failure of *Sensor A* and for the *Test* component, we can derive the probability for the execution of the time-consuming table lookup function and can calculate a probabilistic worst case execution time.

The approach has been evaluated using the tool for failure-dependent timing analysis presented in [Höfig and Domis 2011].

8.4 Efficiently Deploying Safety-Relevant Applications to Integrated Architectures

A method for finding a good deployment (a mapping between the logical and technical viewpoints) has to consider multiple aspects that influence costs and feasibility. One of these aspects is safety, and it is addressed by the approach described in this section. The SPES quality aspect safety contains a two-stepped approach for supporting a safety-related deployment. The goal of the first step is to find a promising deployment candidate using system-level information. The second step investigates the feasibility of the candidate by separately investigating the more detailed safety dependencies between each application and its host platform.

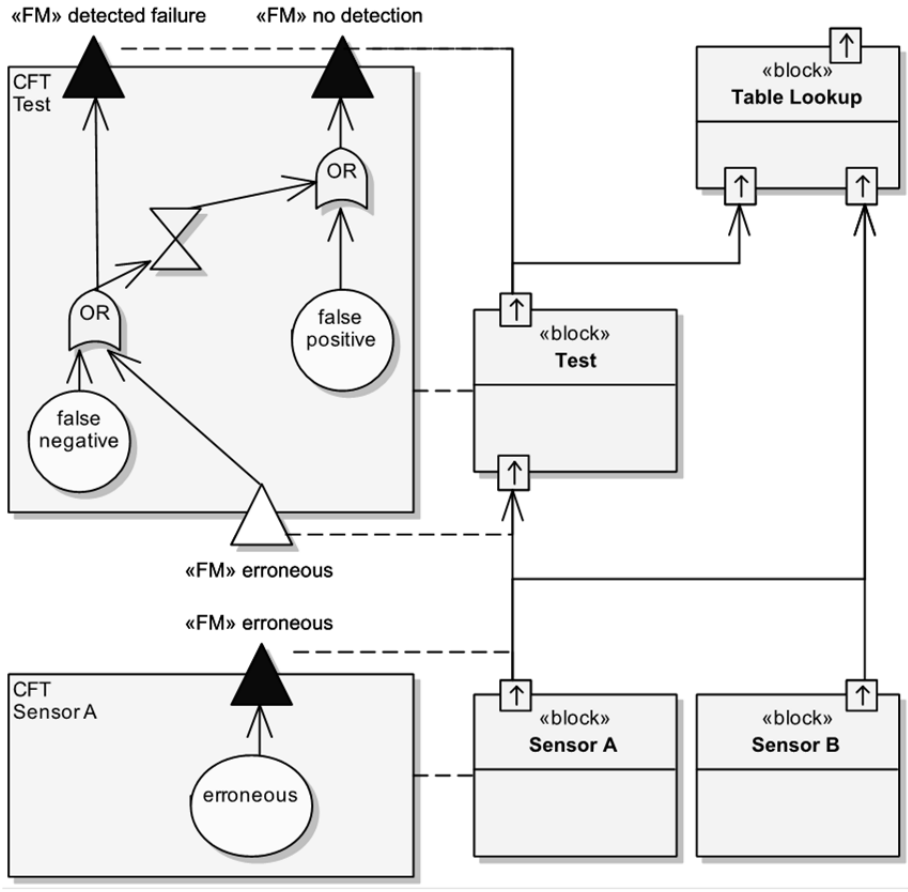


Fig. 8-2 *Sensor example system in SysML with component fault trees*

A deployment starts with a list of applications (high-level logical components), modeled as a network of communicating logical components (as specified in the logical viewpoint introduced in Chapter 6) that have to be deployed onto a set of computer platforms (as specified in the technical viewpoint introduced in Chapter 7), possibly containing several partitions. Fig. 8-3 shows a deployment calculated by a tool developed in the SPES project.

The algorithm for identifying promising candidates uses two metrics to calculate a quantified evaluation of the suitability of a deployment with regard to safety requirements [Zimmer et al. 2012].

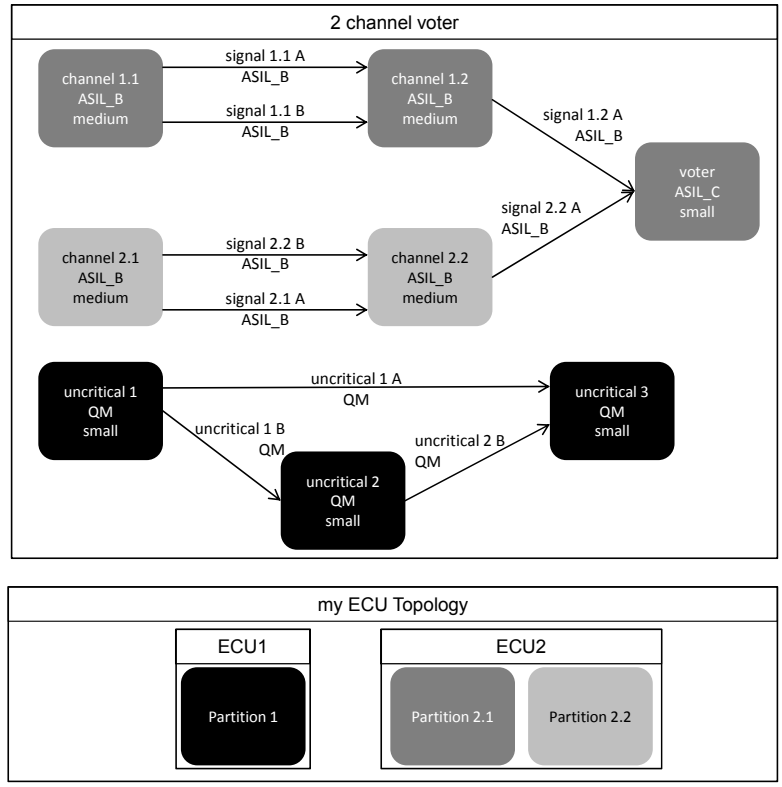


Fig. 8-3 *A network of software components deployed to a computer topology. The deployment of a component to a partition is indicated using the same shade of gray*

The cohesion metric evaluates criticality homogeneity within partitions

The first metric is called the cohesion metric and evaluates the homogeneity of application criticality levels in a partition. As already mentioned, a platform may comprise multiple partitions. Since freedom from interference is not guaranteed within one partition, every application in a partition has to be developed according to the maximum level of criticality of all applications in a partition.

Example 8-2: Cohesion metric

If, in the example shown in Fig. 8-3**,** the component *voter* and the component *channel 1.1* had both been deployed to *partition 2.1*, *channel 1.1* would have to be developed according to ASIL C. The cohesion metric reflects the costs entailed by these criticality increases.

The coupling metric evaluates volume and criticality of safety-relevant communication

The second metric is called the coupling metric and evaluates the volume of safety-relevant communication. If safety-relevant applications residing in different partitions exchange signals, undetected failures in this communication can cause a hazardous outcome. In order to prevent these inadvertent situations, safety mechanisms have to be developed and installed to detect or prevent failures. The coupling metric takes the costs caused by the volume of safety-critical communication, especially in-between platforms, into account.

After these metrics have been used to derive a deployment, the second step of the approach comes into play. The goal of this step is to assist the integrator in checking whether each application software component can run safely on its host platform, and if so, to assist in generating appropriate evidence. The method used in this second step is called VerSaI (*Ver*tical *Sa*fety *I*nterfaces) [Zimmer et al. 2011].

Demands and guarantees specify a contract-like interface

Before the *safety compatibility* between application and platform can be checked, demands and guarantees have to be specified. Demands are typically used to express all the properties an application needs the platform to have in order to be executed safely, whereas the guarantees represent the safety-related properties the platform possesses. A compatibility check is successful if a sound argument for the fulfillment of the demands with the available guarantees can be established. To enable tool-supported integration, the VerSaI approach offers a semiformal language to model these demands and guarantees.

The language comprises a finite number of elements

The language consists of a number of elements each representing a certain type of demand or guarantee exchanged by an application and a platform. This implicates the noteworthy fact that there is a finite number of language elements and, therefore, also a finite number of dependencies expressible with the language. First evaluations have shown that this is suitable because the typical service relationships between an application and a platform are finite and regular too, which is also the reason why it was possible to standardize platform interfaces in the first place.

The VerSaI language is integrated into model-based design artifacts

If the compatibility of an interface specified with the VerSaI language is checked, the demands and guarantees that have a potential relationship have to be identified first. This is done using the integration of the VerSaI language into the SPES modeling framework. A demand about the detection of a signal corruption is, for example, related to the model element representing the signal. On the other hand, a guarantee about detecting signal corruptions is related to the representation of the respective communication channel. If the detailed deployment of the signal to the com-channel is modeled, VerSaI uses this information in a

transitive manner to relate the corresponding demands and guarantees. This principle is depicted in Fig. 8-4.

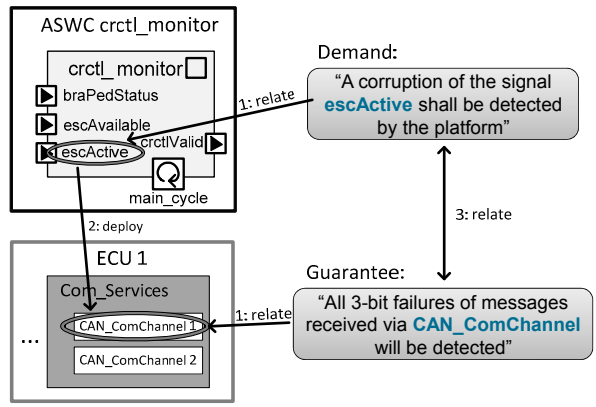


Fig. 8-4 *Relating demands and guarantees using deployment information*

The integration of applications and platforms is supported by a strategy repository

The final step of the method is checking whether each demand can be met with the guarantees identified as relevant in the previous step. In contrast to conventional interfaces, it is usually not possible to simply match the demands and guarantees respectively. In fact, an additional fragment must be generated in the safety case providing the arguments and evidences that the demands of the platform are met by the guarantees given by the platform. To this end, this step is supported by a strategy repository. The repository contains expert strategies that are selected and presented to the integrator, and that describe what guarantees are needed to fulfill the current type of demand and how to generate a piece of evidence containing a sound argument. It is important to note that, despite the formal basis of the language, each language element has a representation in natural language. This allows the human brain to read and evaluate the specification of demands and guarantees and the final argument generated after integration. This argument can be distributed for reviews and assessments.

The two-step approach presented allows the user to calculate a deployment automatically and assists the user in checking and arguing the safety of the chosen deployment.

8.5 Integration in the SPES Modeling Framework

In the SPES modeling framework, safety is represented as a quality aspect that has a crosscutting influence on and is integrated into several viewpoints. This section gives an overview of the integration of the safety quality aspect.

8.5.1 Viewpoints

Relation of the safety quality aspect to the functional, logical, and technical viewpoints

Considering the integration of component-integrated fault trees into the viewpoints of the SPES modeling framework, we have to differentiate between the capability of modularizing and hierarchizing the fault trees that comes with component fault trees, and the capability of integrating the fault trees into a component-based model. Component fault trees, on the one hand, can easily be applied to functions, logical components, and technical components, and thus, to the respective viewpoints as well. However, the component integration part works best with the logical viewpoint since the logical viewpoint metamodel represents communication between components explicitly using input and output ports, and this is best suited for component-integrated fault trees.

Since the deployment models the mapping of logical components to technical components, the methods for efficient deployment of safety-related applications belong to the logical viewpoint as well as to the technical viewpoint.

8.5.2 Abstraction Layers

Relations between the different abstraction layers of a C²FT

The relation between different abstraction layers of a component fault tree is comparable to the relation between different abstraction layers of a logical component as described in Chapter 6. A top-level component fault tree describes the failure behavior of the top-level component and can be decomposed into several lower-level component fault trees describing the failure behavior of the top-level component as an aggregation of several component fault trees.

8.6 References

[Adler et al. 2010] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J.-P. Schwinn, M. Trapp: Integration of component fault trees into the UML. In: Proceedings of 3rd International Workshop on Non-functional Properties in Domain Specific Languages (NFPinDSML2010). DOI: 10.1007/978-3-642-21210-9_30.

[Domis and Trapp 2009] D. Domis, M. Trapp: Component-based abstraction in fault tree analysis. In: Proc. of the International Conference on Computer Safety, Reliability and Security (SAFECOMP 2009). DOI: 10.1007/978-3-642-04468-7_24.

[Domis et al. 2010] D. Domis, K. Höfig, M. Trapp: Consistency check algorithm for component-based refinements of fault trees. In: Proceedings of International Symposium on Software Reliability Engineering , 2010.

[Höfig 2011a] K. Höfig: FDTA – A toolchain for failure-dependent timing analysis. In: Proc.11th International Workshop on Worst-Case Execution Time (WCET) Analysis, 2011.

[Höfig 2011b] K. Höfig: Timing overhead analysis for fault tolerance mechanisms. In: Proc. Zweiter Workshop zur Zukunft der Entwicklung softwareintensiver eigebetteter Systeme (ENVISION2020), LNI Vol. P-184, GI, 2011.

[Höfig and Domis 2011] K. Höfig and D. Domis: Failure-dependent timing analysis. In: Proc. 2nd International ACM Sigsoft Symposium on Architecting Critical Systems, 2011.

[Höfig et al. 2010] K. Höfig, D. Domis, M. Trapp, H. Stallbaum: Pattern-based safety engineering. Semantic enrichment of system architecture models for semi-automated safety analysis. In: Proceedings of European Safety and Reliability Conference, 2010.

[Zimmer et al. 2011] B. Zimmer, S. Bürklen, M. Knoop, J. Höfflinger, M. Trapp: Vertical safety interfaces - improving the efficiency of modular certification. In: Proceedings of the 30th International Conference of Computer Safety, Reliability, and Security, 2011.

[Zimmer et al. 2012] B. Zimmer, M. Trapp, P. Liggesmeyer, J. Höfflinger and S.Bürklen: Safety-focused deployment optimization in open integrated architectures. In: Proceedings of the 31st International Conference of Computer Safety, Reliability and Security, 2012.

Robert Hilbrich
J. Reinier van Kampenhout
Marian Daun
Dr. Thorsten Weyer
Dominik Sojer

9

Modeling Quality Aspects: Real-Time

Timing is an integral part of systems interacting with their context. The correctness of real-time systems depends on logical correctness and proper timing. A model-based engineering approach for real-time systems builds on modeling platform-specific and platform-independent resource requirements as well as resource capabilities. These artifacts can be expressed with extensions to the requirements viewpoint and the technical viewpoint of the SPES modeling framework. Schedulability analysis of the system model can be used to generate a deployment of software tasks to hardware components.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_9,

9.1 Introduction

Timing as a matter of correctness, not only performance

The meaning of real-time computing is overloaded and ambiguous. Often it is associated with a quick and immediate system response to an external event or it is used to describe the performance of multimedia systems that achieve more than 25 frames per second. In embedded systems, real-time computing refers to demanding a reaction to an event in accordance with the progression of time in the system context. The correctness of the system no longer depends solely on the logical correctness of the computation, but also on the point in time at which these results are produced [Buttazzo 2004].

Real-time systems have to be deterministic and predictable

Depending on the consequences if these timing requirements are not met, two classes of real-time systems can be distinguished: hard real-time systems and soft real-time systems. In the case of a hard real-time system, the violation of timing requirements may lead to catastrophic consequences (e.g., harming human life). Soft real-time systems, on the other hand, are less stringent about timeliness. Missing a deadline may affect the performance, but it does not cause serious damage or compromise the system behavior.

While the main objective in high-performance computing is to minimize the average response time for all running tasks, real-time computing is about satisfying timing requirements for all tasks. Especially in hard real-time computing, the system has to exhibit the same timing properties for all tasks under all circumstances. Therefore, these systems have to be fully deterministic and predictable. In order to build a hard real-time system, the progression of real time in the system context—not just logical time—has to be considered during the engineering of the entire system design.

The next section gives a brief overview of model-based real-time engineering and introduces the notions of platform-independent and platform-specific timing requirements. Sections 9.3 and 9.4 then illustrate platform-independent and platform-specific timing requirements respectively in more detail. Section 9.5 illustrates schedulability analyses.

9.2 Model-Based Real-Time Engineering

Real-time engineering incorporates timing requirements and resource capabilities

The engineering principle of strictly separating application development and its deployment on a distributed, technical platform (see Chapter 3) requires a real-time engineering approach that distinguishes between:

- High-level, platform-independent timing requirements
- Platform-specific timing requirements of software tasks
- Platform-specific timing capabilities of hardware resources

An overview of this approach is depicted in Fig. 9-1.

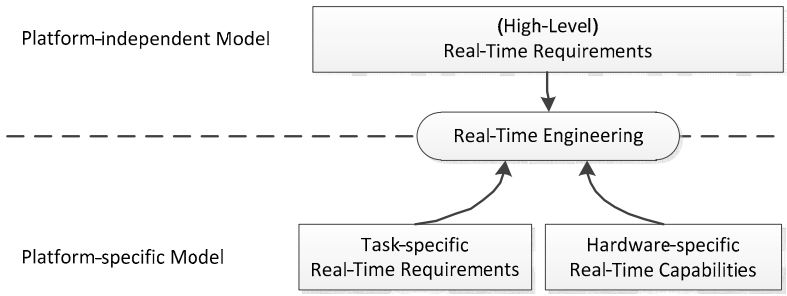


Fig. 9-1: *Real-time engineering has to consider platform-independent properties and platform-specific properties*

The quality aspect real-time is a crosscutting concern in the SPES metamodel

When all requirements and resource capabilities are precisely captured during the engineering process, the major challenge of real-time engineering lies in the search for a spatial and temporal deployment from software tasks to hardware resources, so that all real-time requirements are satisfied. This deployment has to ensure that no resource is overbooked at any point in time, so that nondeterministic waiting times due to a congested resource do not result in missed deadlines. Therefore, the quality aspect real-time was not considered in isolation in the SPES modeling framework. It is a crosscutting concern and affects the requirements viewpoint (see Chapter 4) and the technical viewpoint (see Chapter 7).

9.3 Modeling Platform-Independent Real-Time Requirements

Real-time requirements relate wall-clock time progression to requirements models

Essentially, real-time requirements establish a direct relationship from functional requirements to the progression of wall-clock time. They can be defined explicitly at the beginning of the engineering process as high-level requirements, or they are derived implicitly as platform-specific requirements as a result of a step-by-step refinement process during the system design. This section focuses on the modeling of high-level real-time requirements.

As discussed in Chapter 4, there are different requirements artifact types that describe various aspects of the entire system. These artifacts are generally not built at the same time in the engineering process, but rather in sequential steps. In most cases, real-time properties will not be taken into account in all artifact types. For instance, the structural requirements model will most likely be free from time constraints. On the other hand, goal models, scenario models, and the operational requirements, as well as behavioral requirements models, will have to address time constraints.

9.3.1 Real-Time and Goals

Soft goals may be decomposed into fine-grained hard goals during the development

Considering real-time constraints within requirements engineering has an essential impact on the elicitation of goals and the documentation in goal models. For this purpose, no specific additional modeling element is needed, but every timing constraint has to be documented within the goal model. By documenting these real-time properties, it is reasonable to descend to a more concrete and precise level. There must be a description of exactly which part or process of the system is covered by this time constraint and the maximum processing time allowed must be noted. As it may be difficult to start at such a detailed abstraction layer, abstract goals (called soft goals) may be defined first and decomposed into fine-grained hard goals during the different development steps, considering even scenarios and the functional and behavioral requirements perspectives.

Hint 9-1*: Rules for checking goal models*

- ❑ Have all real-time relevant requirements been considered?
- ❑ Have all real-time requirements been decomposed into fine-grained hard goals allowing for measurement?

9.3.2 Real-Time and Scenarios

Scenarios help to detail goals

Scenarios detail the goals developed earlier. Inversely, each real-time goal should be detailed in at least one scenario. A scenario may be affected by different real-time goals. Hence, it is insufficient to simply relate complete goals and complete scenarios by means of traceability links. The different parts of the scenario have to be related to the different goals. While one major use case of using real-time scenarios is to check real-time requirements for consistency, every part of the single scenario has to be related to a concrete measurable goal. If not, assumptions of real-time behavior have to be made and documented in the scenario (e.g., messages, processing parts). After this has been done, the acceptable runtime for each scenario can be calculated. The ordering structures (e.g., in use cases or in high-level message sequence charts, hMSCs) also have to be related to goals. Therefore, these goals will in general be on higher abstraction layers than the goals that are related to the single scenarios. For a preliminary timing analysis, both runtimes will be compared, with a determination of whether the expected runtime of the scenario is smaller than the accumulated runtime consisting of the expected runtimes of all single parts of the scenario.

Hint 9-2*: Rules for checking scenario models*

- ❑ Have all real-time goals been detailed by at least one scenario?
- ❑ Have all relevant parts of the scenario been connected to the relevant real-time goals?
- ❑ Have all necessary assumptions been made explicit?
- ❑ Has the real-time information in the scenarios been checked against the real-time information in the ordering structure (such as use cases or hMSCs)?

9.3.3 Real-Time and Operational Requirements Models

Operational requirements models are also affected by real-time constraints. In most cases, it will be sufficient to relate each functionality or process to a real-time goal or a selection thereof. In the latter case, the expected runtime will have to be summed up from all real-time goals of the selection.

Hint 9-3*: Rules for checking operational requirements models*

- ❑ Have all functions been related to the relevant real-time goals?
- ❑ Have all real-time goals been considered by at least one function or the overall composition of the system?

9.3.4 Real-Time and Behavioral Requirements Models

Real-time behavior can be modeled with several specification techniques

Real-time properties have a significant impact on the behavioral requirements models. There are a multitude of specification techniques for modeling real-time behavior, for example, Timed Automata [Alur 1999], PTIDES [Zhao et al. 2007], TReqS [Buckl et al. 2010], and Giotto [Henzinger et al. 2003]. They can be chosen as an artifact type for this perspective. The behavioral specification describes the system behavior on a lower abstraction layer. Hence, real-time constraints must be annotated at each state and at each transition; they should be traced back to the goals. Depending on the development project, developing a real-time behavioral specification can allow for additional decomposition of the real-time goals into more finely-grained concrete goals.

The expected system runtime may be checked against the goals formulated and also against the scenarios. Differences between the artifact types will lead to errors during the design, thus any differences occurring represent a proper indicator of necessary requirements reengineering activities concerning all requirements engineering artifacts involved.

Hint 9-4*: Rules for checking behavioral requirements models*

- ❑ Have all real-time goals been considered?
- ❑ Have all assumptions made within the scenario development been taken into account?
- ❑ Have all scenarios been taken into account? Is it possible to run through each scenario by "executing" the requirements specification? Have constraints also been taken into account within these checks?

9.4 Modeling Platform-Specific Real-Time Properties

In addition to the platform-independent real-time requirements within the requirements viewpoint, the platform-specific properties must also be addressed. The following section describes real-time additions to the foundations contained in the technical viewpoint introduced in Chapter 7.

Annotating execution times is not sufficient for complex and distributed real-time systems

9.4.1 Real-Time Requirements for Software Tasks

Each software task has unique real-time requirements that affect its deployment on the hardware platform. Up to this point, the SPES modeling framework introduced tasks that consist of architecture-specific execution times in order to capture real-time requirements. Especially for

complex and distributed software-intensive embedded systems, this is not sufficient and requires an enhancement of the SPES modeling framework. The extension is depicted in Fig. 9-2.

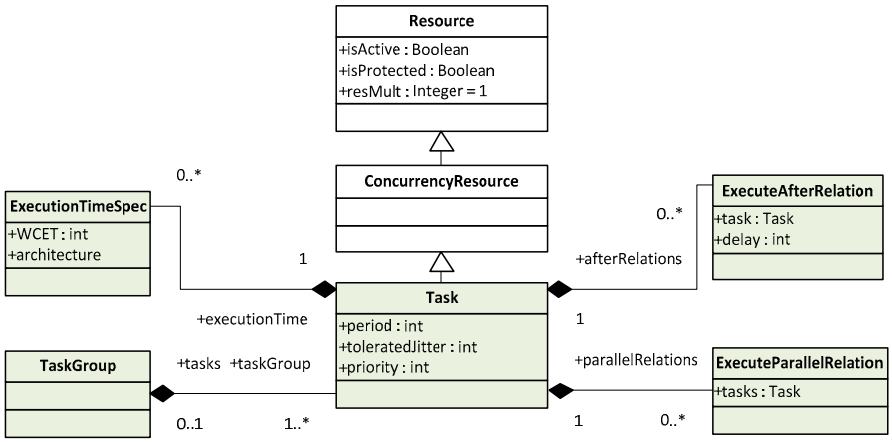


Fig. 9-2 *Modeling software real-time requirements in the SPES modeling framework (elements in light gray represent new or extended components)*

In addition to the specification of a worst case execution time, the tolerated *jitter* and a *period* for synchronous tasks should be modeled. Furthermore, a *priority* can be assigned to a task as well. Priorities can be used to express different criticality levels of software tasks. These are particularly useful when dynamic scheduling techniques based on fixed priorities are used.

"Pipeline behavior" can be specified with additional modeling elements

These attributes focus on real-time requirements for the task in isolation. However, there are also requirements affecting more than just one single task. For instance, when data has to be processed in a pipeline with several stages, a task responsible for a certain stage may have to be executed exactly after completion of the execution of the task for the previous stage. This relationship between tasks can be modeled using the *ExecuteAfterRelation* element.

On a hardware platform offering multiple processing units, e.g., multicore processors, a parallel execution of tasks sharing a common data set is often desirable to optimize cache utilization and increase the system performance. Similar to the *ExecuteAfterRelation* element, the *ExecuteParallelRelation* element allows the specification that a pair of tasks must start their execution at the same time.

Tasks can be grouped to allow co-scheduling

Forcing a set of tasks to execute at the same time is a very stringent requirement. It restricts the search for feasible schedules and may not be

necessary in all use cases. Instead, it is often sufficient to indicate for the scheduler that certain tasks belong to a group. If enough resources are available, the scheduler may then choose to execute these tasks at the same time, thus optimizing the system performance. This approach is commonly known as "co-scheduling" [Ousterhout 1982] or "gang scheduling" [Jette 1997]. In the extension to the SPES modeling framework in Fig. 9-2, tasks can be grouped using the *TaskGroup* element.

9.4.2 Hardware Real-Time Capabilities

In the technical viewpoint (see Chapter 7) of the SPES modeling framework, hardware is modeled as offering resources on which consuming resources such as software tasks can be deployed. The hardware of a system may comprise computation, communication, storage, and I/O resources. A single resource may be shared between multiple consumers. Since computing and communication resources represent the most important hardware components in real-time computing, we mainly focus on them in the following.

Computing resources and communication resources affect a deterministic execution

The basic concept of a *ComputingResource* in the technical viewpoint comprises processing devices that can store and execute tasks. It is a specialized *ProcessingResource*. Modern processors in embedded real-time systems may contain multiple pipelined computational cores comprising ALUs and FPUs. To be able to accurately describe the real-time capabilities, the *HWProcessor* and *CommunicationResource* elements must be extended as shown in Fig. 9-3.

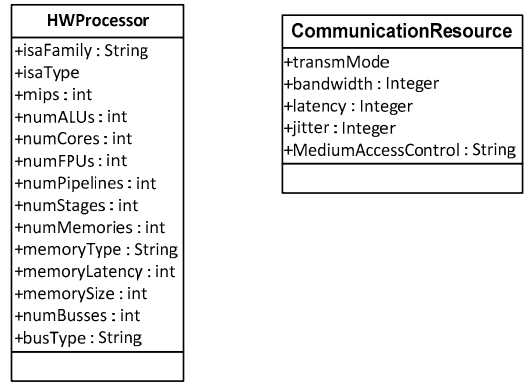


Fig. 9-3 *The extended HWProcessor and CommunicationResource metamodel classes*

The *HWProcessor* class is extended with two important properties: the on-chip memory hierarchy and the configuration of the memory busses. To describe the first property, the attributes *numMemories*, *memoryType*, *memoryLatency,* and *memorySize* are added to the *HWProcessor* class. This is the minimum information that is required to model on-chip memory access times, assuming a Uniform Memory Architecture (UMA). The type of memory can, for instance, be cache or scratchpad; the latency must be given in cycles. The second property documents the access times of off-chip memory and consists of the *numBusses* and *busType* attributes. Bandwidth and latency are inherent to a certain bus type.

The *CommunicationResource* class is extended in a similar way. In this case, the attributes *bandwidth*, *latency*, *jitter,* and *MediumAccessControl* are added. These properties constitute the Quality of Service (QoS) that a communication resource offers. This is an important measure for quantifying its real-time capability.

9.5 Schedulability Analysis

Static schedules for operating systems can be generated automatically

With all real-time requirements and capabilities being precisely captured, schedulability analysis enables an efficient model-based engineering approach. This is especially the case for the construction of static operating system schedules — the core of a time-triggered real-time system [Kopetz 1991]. The use of static schedules on a fixed cyclic basis is a highly recommended design pattern for achieving and guaranteeing a fixed allocation of hardware resources at runtime.

The automated generation of static schedules based on a timing model is a first step towards a model-based engineering of hard real-time systems. In [Hilbrich 2011], such a generator for the avionics domain is described. Within seconds, a static schedule for a fixed time period is created and can be exported to configure a real-time operating system. Whenever a schedule is successfully constructed (see Fig. 9-4), it is guaranteed to satisfy all timing requirements in the model.

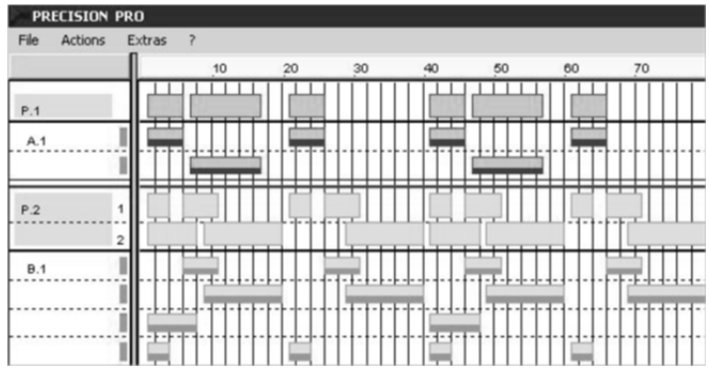


Fig. 9-4 *Graphical output of a generated static schedule*

9.6 References

[Alur 1999] R. Alur: Timed automata. In: Theoretical Computer Science, Vol. 126, No. 2, 1999, pp. 183-235.

[Buckl et al. 2010] C. Buckl, I. Gaponova, M. Geisinger, A. Knoll, E. A. Lee: Model-based specification of timing requirements. In: Proceedings of the tenth ACM International conference on Embedded software. ACM, New York, 2010, pp. 239-248.

[Buttazzo 2004] G. C. Buttazzo: Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications. 2nd Edition, Springer, Santa Clara, CA, USA, 2004, pp. 4-9.

[Henzinger et al. 2003] T. A. Henzinger, B. Horowitz, C. M. Kirsch: Giotto: A time-triggered language for embedded programming. In: Proceedings of the IEEE, Vol. 91, No. 1, 2003. pp. 84-99.

[Hilbrich 2011] R. Hilbrich, H.-J. Goltz: Model-based generation of static schedules for safety critical multi-core systems in the avionics domain. In: Proceeding of the 4th international workshop on Multicore software engineering, Waikiki, Honolulu, HI, USA, 2011, pp. 9-16.

[Jette 1997] Jette, M. A: Performance characteristics of gang scheduling in multiprogrammed environments. In: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM), ACM, 1997, pp. 1-12.

[Kopetz 1991] H. Kopetz: Event-triggered versus time-triggered real-time systems. In: Operating Systems of the 90s and Beyond, Lecture Notes in Computer Science Vol. 563, 1991, pp. 86-101.

[Ousterhout 1982] Ousterhout, J. K.: Scheduling techniques for concurrent systems. In: Proceedings of 3rd International Conf. on Distributed Computing systems, 1982, pp. 22 – 30.

[Zhao et al. 2007] Y. Zhao, E. A. Lee, J. Liu: A programming model for time-synchronized distributed real-time systems. In: Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium, Bellevue, WA, USA 2007. pp. 259-268.

Part III

Application and Evaluation of the SPES Modeling Framework

Dr. Andreas Jedlitschka
Dr. Ulrich Löwen

10

Overview of the SPES Evaluation Strategy

The purpose of this chapter is to introduce the application of the methodologies in the following application domains: automation, automotive, avionic, energy, and healthcare. It describes the evaluation strategy for the systematic evaluation of the SPES modeling framework and presents the process for selecting appropriate case studies, as well as the example phase model that allows a comparison of the case studies. It also explains the common underlying structure of the chapters in Part III of this book.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_10,

10.1 SPES 2020 Evaluation Strategy

Motivation: systematically obtain information on the benefits of the SPES modeling framework

To assess the impact of a technology on certain objectives, such as the technology's inherent benefit or efficiency, systematic evaluations based on scientifically sound criteria have to be planned and conducted. An empirical approach provides the foundation for the ideal research process that allows a meaningful assessment of the results of the evaluation with respect to the objectives of the investigation. Empirical data illustrate the potential and progress associated with the use of the technology. The vision is to demonstrate the benefits of a technology to industry and research through empirically obtained quantitative statistical values and qualitative information.

Evaluation strategy in SPES 2020

The SPES modeling framework was piloted and evaluated in the application domains following a common evaluation strategy. The strategy comprises a set of objectives from research partners and application partners, as well as general methodological guidelines to support planning, conducting, and analyzing empirical studies. The systematic evaluation approach of the SPES modeling framework has its origins in [Chen and Rossi 1983] and uses empirical methods to obtain data to explain or explore phenomena and to derive suggestions for future developments.

Main evaluation device: case studies

The main device used for evaluation in the application domains was representative case studies. The case studies provided a common platform for evaluating the SPES modeling framework by focusing on different aspects of embedded system development. According to [Yin 2003], case studies are "… an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" (p. 13). Often, case studies have small sample sizes and do not allow for controlling confounding variables. Case studies as applied in the application domains are categorized as feasibility studies. The focus of most of the case studies was on evaluating whether the SPES modeling framework can be applied in the domain, whether the viewpoints are seamlessly integrated, whether the implemented tools supported the approach, and on identifying improvement potential. In several studies, the concept of semistructured feedback sessions was used to obtain feedback from the users. The results of a case study are specifically valid in the environment they were obtained from. Case studies must therefore be selected with special regard to the evaluation focus. Section 10.2

summarizes the selection process for the case studies in Chapters 11 through 15.

Supporting means for evaluation: experiments

Experiments were used as supportive means for the evaluation and were intended to measure and analyze the effect of systematic variations in the independent variable on the dependent variable. Often, an experiment included an experimental group (in which "treatment" was applied) and a control group (control/no treatment applied) to show the effect of the "treatment" between the groups, i.e., the effects of applying the SPES modeling framework ("treatment" in the sense of above) in a development setting was compared to the method currently used (no treatment) with regard to a certain quality focus. The aim of experiments is to provide statistically representative results that are valid for the population from which the sample is drawn. For further details on several experimental designs, we refer to [Shadish et al. 2002].

10.2 Selecting Appropriate Case Studies

Embedded systems in the various application domains have different focus areas and are subject to different development approaches. Therefore, it was necessary to select a number of representative case studies within each application domain that would accurately reflect the special features of the domain, as well as present a good example of the challenges that the individual application domains face during development. This was necessary to ensure meaningful results from the evaluation of the SPES modeling framework.

Agreeing on a development phase model

Due to the heterogeneous nature of embedded systems (cf. Chapter 1) and their development (cf. Chapter 2), an application domain-independent phase model of the development of embedded systems was established. This phase model consists of the following development phases:

- Product definition
- System definition (Requirements)
- System definition (Architecture)
- Device definition (Requirements)
- Device definition (Architecture)
- Software and Hardware development as well as other development disciplines
- Integration

These development phases are common in most application domains. On this basis, a fundamental understanding across application areas for a

universal, model-based development process of embedded systems was created, allowing for common conclusions about the SPES modeling framework from the evaluation activities in the individual application domains (see Chapters 11 through 15).

Matching phase model, evaluation goals, and case studies

In addition, challenges and/or evaluation goals of individual application domains were identified and case studies were defined in each application domain. Particular attention was paid to ensuring that the case studies adequately addressed the engineering challenges of the application domains (cf. Chapter 2). Fig. 10-1 shows the relation of the case studies identified and the respective development phases they pertain to.

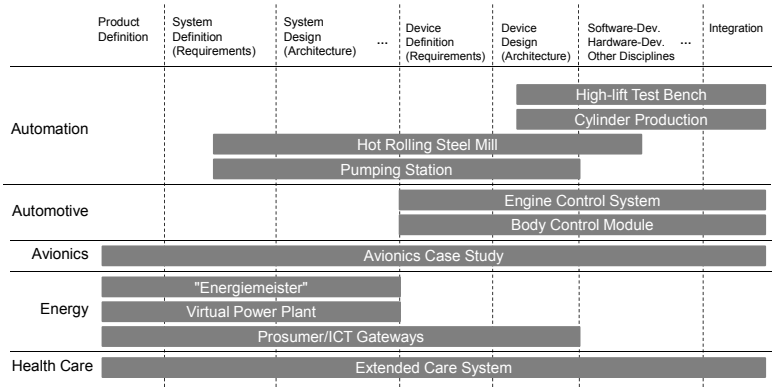


Fig. 10-1 *Classification of the SPES 2020 case studies in the development phases of embedded systems*

10.3 Structure of the Following Chapters

Chapters 11 through 15 focus on the use of the SPES modeling framework in the five application domains. To give the reader a better understanding and a simple cross-comparison between the domains, a uniform structuring concept consisting of five sections each was selected.

Section 1: Application domain overview

- The first section gives an overview of the application domain. In addition to providing a description, it also discusses the economic relevance, such as the proportion of the application domain in relation to the gross social product or details on added value. From a technical standpoint, the first section explains where embedded systems will be used today and in the future and the level of relevance they have. In addition to characteristics such as quantities and costs, it also explains life cycle requirements and general product and/or system properties. If any specific quality

considerations of the system under development (SUD) must be accounted for, e.g., safety properties, real-time response, reliability, or performance, the first section elaborates on these considerations. To conclude, the section discusses the particular problems of the application domain with regard to embedded systems.

- Section two of each chapter discusses the strategy used to evaluate the SPES modeling framework within the application domain. It unveils the central questions that had to be answered in the evaluation and explains how the problems described in the first section can be solved. This section also examines the procedure used for the evaluation and how the academic and industrial partners cooperated.

Section 2: Application domain-specific evaluation strategy

- The third section gives an overview of all evaluation activities, outlining the achieved results and integrating them into the SPES modeling framework.

Section 3: Evaluation activity summary

- A selective, but more in-depth discussion of a specific evaluation takes place in the fourth section. This section discusses the content-related objective and explains the specific activities and methods used in detail.

Section 4: In-depth discussion of selected activities and results

- Section five summarizes the results and answers the question as to what was achieved or not achieved within the scope of the evaluation. It presents suggestions for additional scientific work on the topic, along with the knowledge gained.

Section 5: Summary and conclusion

10.4 References

[Chen and Rossi 1983] H. T. Chen, P. H. Rossi: Evaluating with sense: The theory-driven approach. Evaluation Review, Vol. 7, No. 3, 1983, pp. 283-302.

[Shadish et al. 2002] W. R. Shadish, T. D. Cook, D. T. Campbell: Experimental and quasi-experimental design for generalized causal inference. Houghton-Mifflin, Boston, 2002.

[Yin 2003] R. K. Yin: Case study research. Design and methods. 3rd Edition, Sage, London, 2003.

Dr. Thomas Wagner
Dr. Jan Christoph Wehrstedt
Dr. Ulrich Löwen
Tobias Jäger
Prof. Dr.-Ing. Alexander Fay
Peter Schuller

11

Application and Evaluation in the Automation Domain

The main focus of our work in the automation domain was on the scientific basis and technical implementation of the interaction and the integration in classically heterogeneous systems and development landscapes. This comprised the integration of automation devices and system components, activities, and models of various engineering disciplines, as well as simulation and validation within the scope of a model-based, quality-assured engineering process. The SPES modeling framework formed the foundation for all developments and was extensively evaluated based on multiple real-world case studies in industry. Results show a significant contribution toward the consolidation of domain-specific modeling and systems.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_11, © Springer-Verlag Berlin Heidelberg 2012

11.1 Overview: Application Domain Automation

Purpose and importance of automation engineering

Whether in the manufacturing or processing industry, in mechanical engineering, in transportation, or in logistics — automation engineering plays a key role in controlling and structuring complex systems. Automation engineering describes the design and realization of automated systems for technical processes such as steel production, manufacturing, or refining. It includes process measuring, open and closed loop control, monitoring, alarming, locking, and optimization.

Embedded systems: a core asset in automation

Embedded systems are an essential asset for automating and controlling these processes within industrial solutions and plants. They comprise electrical or electronic devices for measuring and influencing process parameters, hardware components, a communication system, and software for the different automation functions. Embedded systems are used in many different forms reflecting the specific characteristics of processes or industries. With regard to applications of embedded systems in automation, it is important to differentiate between individual devices or single systems, which are developed as mass products, and systems of systems in the form of machines and industrial plants, which are individually implemented in implementation projects for specific customer requirements (Example 11-1). Hence, the same (from an internal point of view) devices or systems deployed in two different systems of systems will appear totally different (from an external point of view) due to the configuration, customization, and integration in applications of different domains with different requirements and constraints.

Example 11-1: *Application areas of embedded systems in automation*

- **Automation devices**: Generally, standardized, configurable, or freely-programmable hardware components
- **Machines and apparatuses**: Functional, independent units such as robots, machine tools, or process cells, partly prefabricated and partly fitted
- **Industrial systems**: Individual configuration of machines, apparatuses, and automation devices for implementing a spatially dispersed process, e.g., power plant, mill, pumping station, factory

Programmable logic controllers (PLC) or comparable specialized microcontrollers are deployed as hardware. These are electronic control systems used to automate a variety of technical processes featuring multiple input and output arrangements, real-time signal processing,

deterministic logic execution, modular hardware configurability, minimized space, and extended temperature ranges. PLCs are programmed via an interconnected computer using domain-specific standardized programming languages.

Integration of embedded systems is the key challenge in automation

Automation engineering is the key for being able to manufacture top-quality and affordable products in high-wage countries. However, embedded systems in automation engineering solutions require the adaption and integration of intricately connected mechanical, electrical, and IT units from functional, logical, technical, and spatial aspects (see Example 11-2). Conversely, there are stringent requirements for the ability of the individual automation devices and systems to be combined and integrated.

Example 11-2: *Characteristics of embedded systems in automation*

- Specific combination of software and system components from various manufacturers that are partially proprietary and partially comprised of old and new components
- Collaboration and integration of the models of many disciplines
- Integration and maintenance of existing software and system components with new systems and technology changes in the life cycle

Integration must be driven by models

The control of this integration complexity requires effective collaboration across the domain. In this case, the software takes an increasingly important role because it forms the last link during the implementation or initial operation of the entire system and thus, must ensure the proper interaction of all disciplines. Moreover, the automation system can only be fully tested in conjunction with the technical process, i.e., following its implementation on-site. An early integration, therefore, can only occur on the model level. However, currently, the individual models of the disciplines are still being developed largely in isolation. These models must be interlinked, whereby it is necessary to ensure consistency throughout the individual disciplines, the entire life cycle of the automation solution, as well as throughout the various levels of automation.

Model-based development is not only expected to improve efficiency in the interdisciplinary development process of automation engineering through networking and parallelization; in addition, appropriately connected models can be used tactically to make requirements and design decisions transparent across all disciplines at an early stage. They can also be used to secure performance parameters of the automation solution early in the proceedings and thus, to minimize development risks and costs [Wagner and Löwen 2010].

11.2 Evaluation Strategy for the Domain

Ongoing coordination of modeling

From the perspective of automation engineering, a promising approach was to start comparing the terms and concepts already proven in the domain with the SPES modeling framework in order to lay a scientific foundation. Therefore, the characteristics and cross-references of discipline-specific models were analyzed at the beginning of the project through the analysis of case studies and actual automation projects of industry partners. The requirements of the domain for an integrated, model-based development process were also determined. In this context, particular focus was placed on the alignment of the various terminologies and model concepts from software and automation engineering, and thus the foundation for the transferability of requirements and solutions was created.

Development-accompanying evaluation of the metamodel in the domain

The concepts of the SPES modeling framework were compared to existing industrial approaches and optimized parallel to the development process. For the evaluation, automation system models were assigned to viewpoints of the SPES modeling framework and integrated in automation engineering development processes. The concepts and methods of the SPES modeling framework were represented through prototypical modeling of selected case studies and tools with the participation of domain experts and the academic partners.

11.3 Overview of Activities and Results

In this section, a general overview of the evaluation context and the conducted evaluation activities is given.

Tab. 11-1 *Industries of the automation domain and SPES case studies*

Area	Explanation	SPES Case Studies
Process plants	Automation of physical or chemical transformations of substances, materials, or energy by a sequence of continuous flow processes	Pumping station plant for water distribution (Siemens AG)
		Hot rolling steel mill (Siemens AG)
Manufacturing plants	Automation of procedures acting upon the forming, working, and joining of materials or items by a disconnected transition of discrete process operations	Test bench for wings (E4You)
		Train control system (Siemens AG)
		Cylinder production (Siemens AG)

11.3.1 Overall Context of the Activities

Best practice modeling concepts for all industries of the automation domain

The activities focused on the development and transfer of best practice methods between the modeling of automation systems and the SPES modeling framework. Multiple complementary case studies from various application areas of the domain (Tab. 11-1) were used to evaluate the SPES modeling framework. In addition, domain-specific characteristics and enhancements of the metamodels were developed and implemented.

11.3.2 Model-Driven Development of Automation Devices

The focus was on the development of best practices and guidelines for the systematic introduction of model-based processes [Fieber et al. 2009]. Moreover, methods for assuring the quality of models in the industrial environment were developed, and the software development for automation devices was observed in particular in these methods [Arendt et al. 2011]. The SPES modeling framework was tested for its applicability on the basis of the case study *Train control system*. In order to deliver proof of the usability of the results, a prototype for the automated quality assurance of models was implemented for the case study and has already been successfully applied in further projects of Siemens AG [Arendt and Taentzer 2012].

11.3.3 Model-Driven Integration and Simulation of Embedded Systems in Industrial Systems

Here, the focus was on the application of the SPES modeling framework for the interdisciplinary modeling (engineering, electrical engineering, software) of industry systems with regard to the integration and dependencies between the different disciplines involved in automation engineering [Jäger et al. 2011], and with regard to the use of models for the improvement of the interdisciplinary collaboration (cf. [Wagner et al. 2011, Fay et al. 2011]). An additional emphasis was on the validation of the embedded systems in systems through the use of functional, logical, and technical models of the systems for simulations [Wehrstedt et al. 2011].

The application of the SPES modeling framework to the description methods of automation engineering and its applicability were evaluated on the basis of three case studies: *Hot rolling steel mill*, *Pumping station*, and *Cylinder production*. Aside from expert analysis, the prototypical re-modeling of actual systems was applied in the studies for evaluation [Lüder et al. 2010, Wagner et al. 2011]. The activities were accompanied

by practical demonstrations. The results of the evaluation are described in further detail in Section 11.4.

11.3.4 Data Models for Automation Runtime Platforms

At the center of our work were the design and the implementation of a runtime platform to which various automation devices and development tools could be connected. Therefore, PLCs (e.g., radCASE), process visualizations (e.g., embedded graphic XiBase9), and test environments (e.g., YAVE) were integrated using the Embedded4You middleware GAMMA on the basis of a central modeling approach.

This data model represents the SPES modeling framework completely and abstracts the system components of embedded platforms, such as hardware, I/O structures, operating systems, and communication. It describes the transition from platform-independent (PIM) to platform-specific data models (PSM). A PIM describes the model of the logical viewpoint in the SPES modeling framework, whereas a PSM is a model of the technical viewpoint of the SPES modeling framework. These models are connected via a mapping relationship. The models of the various abstraction layers and viewpoints require execution semantics to be able to validate properties of the system in early stages of development. Therefore, the modeling approach also contains platform-specific artifacts for the execution, such as process and I/O variables, temporal processes, or memory dependencies. Together with the test environment, the data-centric model enables simulation models to be tested with MATLAB/Simulink and software or hardware in the loop.

The results are applied in diverse embedded platforms and test systems. Special platforms were implemented to prove the modeling on typical hardware environments of automation. The evaluation occurred in the *High-lift test bench* case study (modeling of the high-lift test of wings). In that context, a flexible microTCA hardware platform was built. The functional scope of the case study is scalable to modeling in automation.

The proof of concept was shown by implementing and testing the runtime platform for the case study *High-lift test bench*. The technical implementation of the model only occurred in the form of an example within the scope of the project due to high costs. As far as the development environment is concerned, the most significant concepts were developed and implemented using prototypes. The activities were accompanied by scientific work for modeling temporal aspects and the introduction of the results into the standard [VDI/VDE 2657].

11.4 Application and Evaluation of the SPES Modeling Framework

Collaboration of scientific and industrial partners

The models and results presented in the following consolidate the work from numerous case studies that were jointly developed by Siemens AG, the Helmut Schmidt University/University of the Federal Armed Forces Hamburg, OFFIS e.V., the University of Duisburg-Essen, and the Technische Universität München. For the purposes of clarity and confidentiality of real project information, they will be explained based on the simplified demonstration model of a production technology system.

11.4.1 Domain-Specific Challenges

Software within automation systems is becoming more complex and crucial to such systems [Achatz and Löwen 2005], resulting in some important domain-specific challenges — this was the focus of the evaluation.

Integration of multiple engineering disciplines

A variety of engineering disciplines are involved in the realization of an automation system: electronics, mechanics, process technology, embedded hardware and software. The disciplines all require their own models and methods that are specialized for their particular design purpose. However, design decisions and results of different disciplines depend on each other. A modeling approach has to offer system models that span disciplines, as well as views onto these models that support discipline-specific methods and models and ensure the consistency and traceability of the data of the different views.

Scalability for large number of entities

Complexity also arises from the great number of functions, signals, and devices in automation systems. Models, and especially tools, have to stay in control and must support the engineers to stay on top of things.

Process integration of modeling approach

Due to their complexity, in practice, automation projects are process-driven and consist of different phases in which the projected system is designed, refined, and made more specific step-by-step. Important phases are concept engineering (for bid preparation), basic engineering, detailed engineering, and installation and commissioning. In each phase, different objectives and requirements have to be supported by models (see Tab. 12-2). For example, a project must be able to estimate the major cost items as early as possible for the purposes of bidding (e.g., equipment, engineering and construction). A modeling approach must allow this information to be revealed without excessive effort.

Tab. 11-2 *Typical phases of an automation engineering project*

Phase	Concept Engineering	Basic Engineering	Detailed Engineering	Installation & Commissioning
Aim	Plant is feasible	Plant to be contracted	Plant can be built	Production can start
Typical Tasks	1. Clarify technical scope 2. Analyze requirements & risks 3. Design solution concept 4. Assess effort & quantities	1. Design technical process, plant architecture, and construction 2. Initial technical specifications & calculations 3. Simulation and validation	1. Design all systems 2. Fully specify equipment 3. Purchase, manufacture, and implement systems and software	1. Assemble mechanical, electronic, and automation parts 2. Deploy software 3. Check, inspect, and test every operational component

11.4.2 Introduction to the Case Studies

Case study: Cylinder machining unit

The demonstrator represents a processing cell in a production system for manufacturing cylinder heads and offers the possibility of mapping and simulating planning processes and models of actual industrial systems. The reduced complexity of the demonstrator allows representative results to be achieved in a short time, and these results are then transferable to the actual planning process for industrial systems.

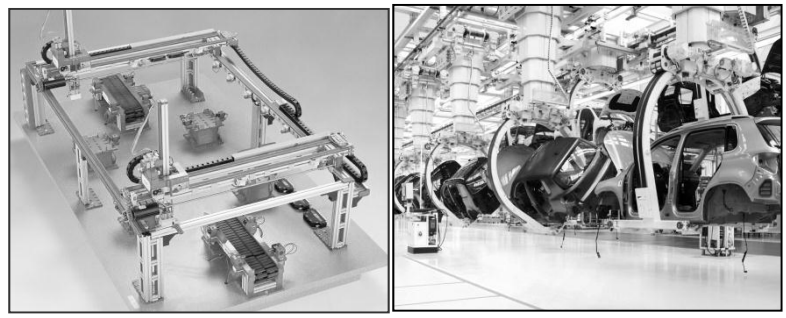


Fig. 11-1 *Demonstration model and example of a production system*

The demonstrator consists of four stations for work piece processing: two conveyor belts and two gantry cranes for transporting work pieces. Automation enables multiple work pieces in the cell to be processed concurrently. To achieve this, optical sensors for determining the work piece position are installed, as well as push buttons for determining the crane position. For the automation, a PLC S7-400 is used, and is connected via PROFIBUS to a total of 52 sensors (position measuring, work piece identification) and actuators (drives).

The system was engineered using plant engineering tools COMOS, Mechatronics Concept Designer, and the automation system SIMATIC PCS7. In addition to the configuration of the hardware and communication, this engineering also included the creation of a control program and the configuration of the operator control panel.

11.4.3 Mapping of the SPES Modeling Framework to the Concept and Description Methods for Automation

4-tier metamodel classifies all entities in automation modeling

To make the SPES modeling framework compatible for modeling automation engineering, the common entity classes, views, relationships, and concepts of the domain-specific models have to be assigned to those of the SPES modeling framework. To enable this assignment independent of the various discipline-specific and industry-specific modeling languages, a generic metamodel of automation engineering was developed (see Fig. 11-2) [Strube et al. 2010]. It describes all elements to be modeled and their relationships relevant from an automation engineering perspective for the functionality and integration of a system. The 4-tier model can be used to interlink the SPES modeling framework to the domain-specific modeling languages. In this context, tiers 3 and 4 represent the automation system. The modeling of the system environment on tiers 1 and 2 is of particular importance, not only for compiling the requirements from the engineering process and resources to the automation, but also for validating all requirements. This requires an overall analysis of the system considering the interaction of processes, resources, hardware, and software.

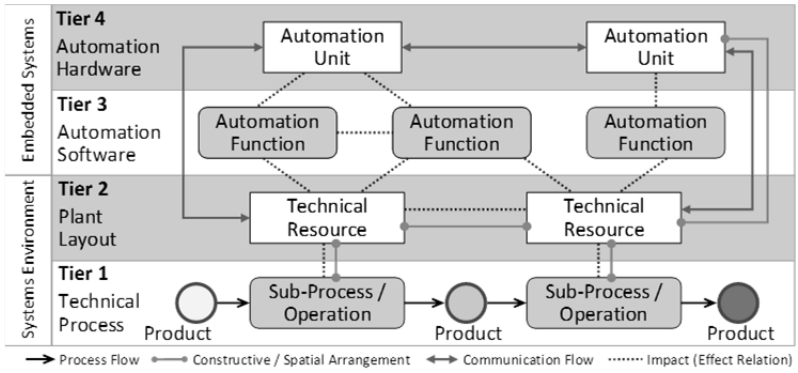


Fig. 11-2 *4-tier metamodel for automated plants [Strube et al. 2010]*

Metalevel domain compatibility: SPES viewpoints cover all entity classes of automation models

Tab. 11-3 shows the assignment of the entity classes of automation models to the viewpoints of the SPES modeling framework as well as the associated domain-specific modeling languages. There is no one to one assignment of the domain-specific modeling languages to the viewpoints of the SPES modeling framework. Many automation engineering models contain functional as well as logical and technical aspects, e.g., the system layout or the process and instrumentation diagram. The requirements viewpoint is not listed because it is still minimally established in the domain and the SPES concepts can be used without further adaptation.

Tab. 11-3 *Mapping of SPES viewpoints to automation entity classes*

Viewpoint	Automation Entity Class		Domain-Specific Models
Functional Viewpoint	Tier 1: Technical process	Sequence of process operations with which raw materials are transformed into final products	Basic/process flow diagram Bill of operations (BOO) Functional architecture
Logical Viewpoint	Tier 2: Plant layout	All types of technical resources (machines, apparatuses) for executing the technical process	Plant breakdown structure Plant layout (CAD model) Material flow diagram Process & instrumentation diagr.
	Tier 3: Automation software	Necessary automation functionality for controlling the process of tier 1 with the resources of tier 2	Process control hierarchy Measuring/set point list State chart/Petri net/SysML Function charts/blocks
Technical Viewpoint	Tier 4: Automation hardware	All devices for controlling and monitoring of the technical process by executing automation software	Parts list: drives, instruments Hardware/network configuration Input/output signal list Wiring & mounting diagrams

Models in automation have a strong physical connection

Only a limited number of technological processes are available for particular process operations, which in turn affect the type and features of technical resources and automation functionality. From the bill of operations of the cylinder processing cell presented in Fig. 11-3, it is obvious that the individual process steps can only be executed by certain types of machines. Further technical requirements arise through the given process templates and the parameters of process operations (e.g., temperature, forces, velocity, geometry). For example, in the case of simultaneous movement of the gantry cranes in the cylinder machining unit, collisions must be avoided. These types of system requirements are modeled according to the SPES modeling framework as contracts (Fig. 11-3), and can therefore be taken into consideration in the development process so that future integration or functional problems can be minimized.

The logical viewpoint serves as a platform-independent description of the logical system architecture. In automation engineering, this can be

mapped to the design of the system layout and the necessary automation functionality according to the process flow (see Tab. 11-3).

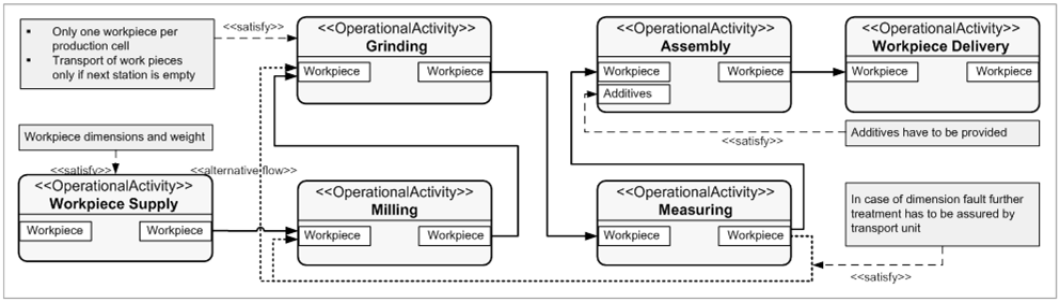


Fig. 11-3 *Functional view: Bill of operations of the cylinder machining unit*

Logical viewpoint models shaped by technical platform

Evaluation showed that, in contrast to the SPES modeling framework approach of a platform-independent logical system model, the layout of automation systems (tier 2) must consider the technological platform to some extent. For example, the type of technical resources needed according to the selected technological process for a process operation must be determined, which in turn also significantly affects the type and features of the automation functionality (tier 3). For example, based on the decision that gantry cranes have to be used to transport work pieces in the system layout of the processing cell instead of robot arms (Fig. 11-4), the number and type of sensors and actuators, and thus, the process signals as well as the type of automation software (e.g., the logical process control instead of numerical movement control) are determined. The automation functionality cannot be modeled independently of the platform because it is dependent on the features of the hardware (temporal relationship, bus systems) and the sensor/actuator interfaces. For the logical viewpoint, therefore, few independent models currently exist in automation engineering; rather the elements of the logical view are primarily described in technology-oriented models.

Due to the fact that the technological process forms the common basis for all disciplines involved in the planning process and their models, the dependencies between the process, the technical resources, and the automation system must also emerge from the functional and logical system model. On one hand, this approach increases the degree of the overall description of the system and in the process, also helps to avoid misunderstandings and integration issues between the disciplines. The challenge from the perspective of automation engineering was in finding suitable modeling strategies so that, on one hand traceability beyond viewpoints exists, and on the other, a viewpoint is closed. For

this reason, the 4-tier metamodel also contains a classification of technically relevant relationships between the entities (Hint 11-1). Furthermore, a method for analyzing and modeling technical dependencies when engineering a system was developed [Jäger et al. 2011, Strube et al. 2011].

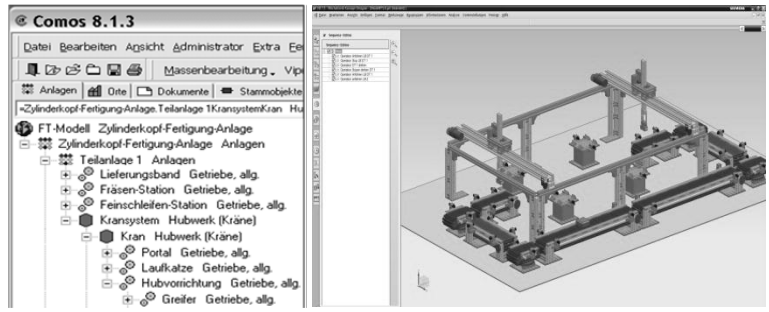


Fig. 11-4 *Logical view: plant breakdown structure modeled in COMOS (left) and 3D layout in Mechatronics Concept Designer (right)*

Functional and logical design is dependent on the technical aspects

Hint 11-1: *Classification of model relations of automation entities*

- **Process flow** represents the transport of items, substances, or energy between process operations and/or storage
- **Structural and spatial associations** represent the spatial or constructive assembly of entities in the plant (e.g., shaft, screw) or mechanical connections for transporting material or substance (e.g., pipeline, cable)
- **Communication relations** between units represent the exchange of signals/information (test value, set point, function call, etc.), e.g., wire connection between sensors and actuators of a technical resource and an automation device.
- **Relationships of effect** represent functional dependencies between technical resources as well as between automation functionality with the controlled resource and the process operation executed

Structural and communication relations are modeled using the SPES modeling framework concepts *logical component, port, connection, mapping,* and *realization*. To model simple flow and relationships of effect, the concepts *allocation, mapping,* and *contracts* are applied. Fig. 11-5 shows this in the example of the logical model of the automation functionalities of the cylinder processing station. The automation functionalities on the next abstraction layer are hardened through the domain-specific modeling languages *continuous* or *sequential function charts,* and thus the transition for the implementation of the logical

components is created with logical PLC programming languages (function blocks), as shown in Fig. 11-6.

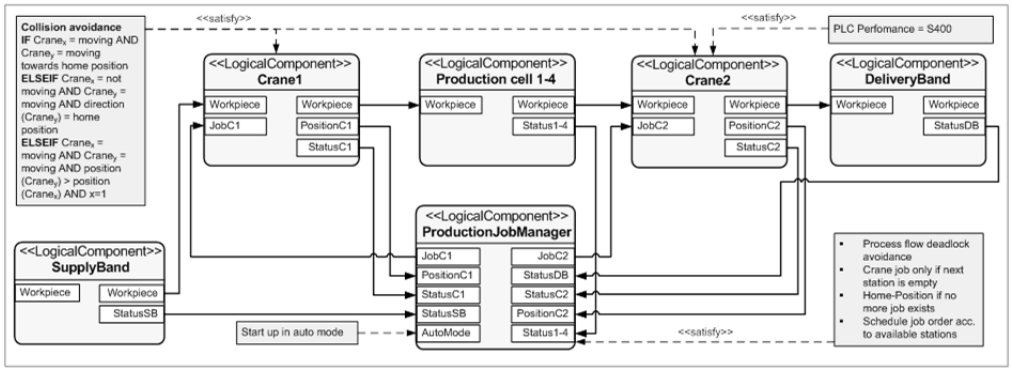


Fig. 11-5 *Logical view of a high abstraction layer: functions, interaction, and contracts*

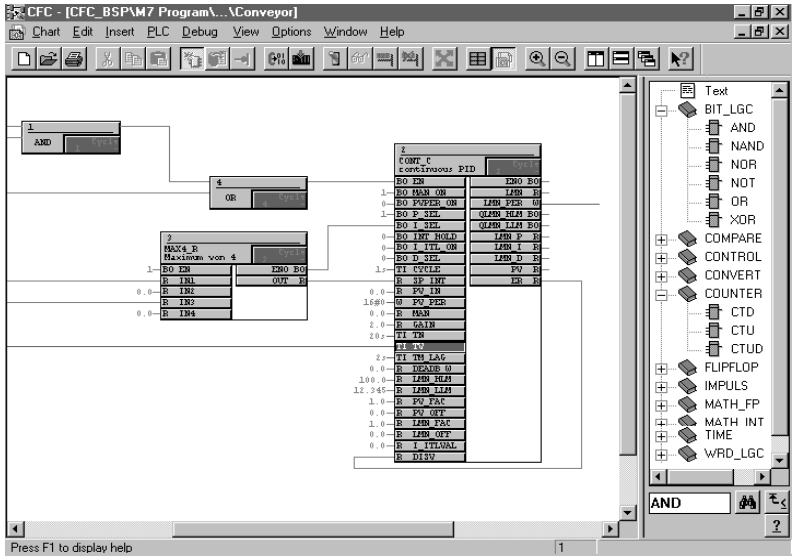


Fig. 11-6 *Logical view of a lower abstraction layer: crane control: continuous function chart*

Modeling of relations of automation entities

Within the technical viewpoint, the logical components of the logical model are allocated to equipment and the software is distributed to the automation devices. The main entities are technical components such as electronic control units, for example, PLC, control cabinet, operator station, and systems, as well as actuators and sensors (Fig. 11-7). This network represents the hardware platform to which the logical

components must be allocated (Fig. 11-8). Automation hardware is typically selected from catalogs and the software is implemented with appropriate tools specific to the hardware. The specific machines and apparatuses are selected for the technical resources. They include additional technical components for discipline-specific subsystems such as power supply, hydraulics, etc.

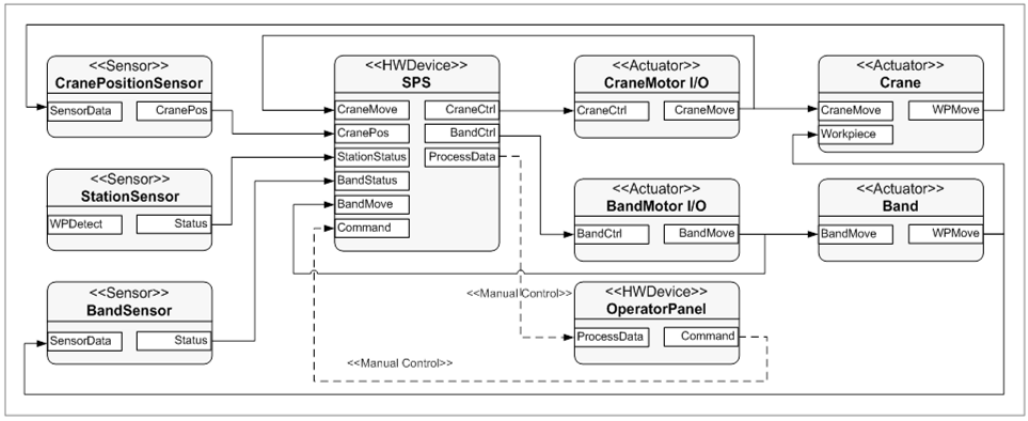


Fig. 11-7 *Technical view: hardware layout*

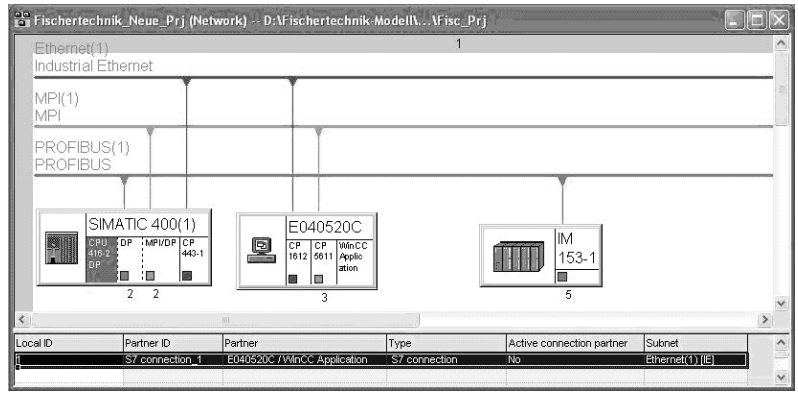


Fig. 11-8 *Technical view: network layout in SIMATIC NetPro*

Through the formalization of the process conditions and relationships of effect, both can be analyzed automatically. In this way, it is possible to determine, for example, whether logical components that depend on each other are also compatible with each other. In order to enable a complete review of the functionality and of the embedded system during technical process control prior to the actual systems deployment, solution concepts have been developed. For example, executable simulation models for

process and automation can be generated from the interdisciplinary complete model [Wehrstedt et al. 2011]. These are used to validate the functional and logical models of the system with the use of simulation tools — both in an early phase as well as during the virtual initial operation. For example, it is necessary for the cylinder machining unit to test whether the control operates without hindrance or collision.

11.4.4 Evaluation of the Abstract Modeling Concepts

In addition to the application of case studies, the modeling concepts of the SPES modeling framework were compared to those of common design methods, modeling languages, and tools of automation. The comparison was based on a criteria catalog with defined and weighted instances (including examples) of conceptual abstractions. The validity and completeness of the criteria catalog for analysis and comparison of modeling and tooling concepts has been proven in more than 25 evaluations for several Siemens business units in recent years.

An overview of the results is presented in Tab. 11-4. In summary, a complete consideration and specification of the concepts *views, hierarchy,* and *abstraction* can be found in the SPES modeling framework. It also became evident that the modeling concepts *modularization, viewpoints, aspects,* and *dependency/mapping* were not yet completely specified in both SPES and automation engineering. The concepts *reuse* and *mechatronics* are only applied in the automation domain and are not currently considered in the SPES modeling framework.

11.4.5 Methodological Approach for the Engineering of Systems on the Basis of the SPES Modeling Framework

The SPES modeling framework is an architecture model and does not provide a process model. Due to the fact that automation projects are strongly driven by processes, a suitable approach for the engineering of automation systems based on the SPES modeling framework was derived. In the SPES modeling framework, the engineering phases *concept, basis,* and *detailed engineering* are supported by the viewpoints. The installation and commissioning phase of standard process models is currently not provided. The models to be created in this phase, however, do not require currently unknown modeling concepts and it was therefore possible to depict them in the SPES modeling framework (Fig. 11-9 left).

Tab. 11-4 *Evaluation of abstract modeling concepts*

Concept	Emphasis		Explanations
	SPES	AUT	
Hierarchy	Very High	Very High	Hierarchization through composition and decomposition is used in both domains. There is a common understanding in the areas of consideration and specification.
Abstraction	High	High	Is substantial in the SPES modeling framework, although the type of abstraction is not specified further. For example, the role concept significant for the automation can be depicted with a functional viewpoint and implementation relationship, though an explicit type of artifact is missing: *role* or *abstract component*.
Modularization	Med.	High	Modularization has to be applied in the SPES modeling framework when hierarchizing. However, the consideration and specification of a hierarchy-wide modularization, such as is implemented in the automation domain with the aid of the group concept (logical areas), is lacking. This would correlate to a viewpoints and cross-aspect viewing aggregation.
Discipline-specific views	Med.	Med.	The viewing concept is implemented in the SPES modeling framework entirely. The discipline-specific design, however, is not sufficiently supported. Aspects allow the internal classification of information within a component, though not an explicit component-wide classification or summary of individual aspects, interfaces, and connections.
Dependency	Low	Low	The mapping concept with the characteristics *implementation, allocation,* and *link to external data* is currently not fully specified. For example, extensions for the representation of complex dependencies with class, features, and functionality are required.
Reuse	None	High	The SPES modeling framework does not currently provide a reuse concept. Libraries, instance, and inheritance are required to support the reuse of engineering artifacts.
Mechatronics	None	Med.	Artifacts are classified in the SPES modeling framework. However, in automation, an artifact frequently comprises information from multiple viewpoints that must be considered cohesively because a significant added value of the modeling arises.

In automation engineering, firstly the system requirements are analyzed (decomposition). As explained by means of the case study, requirements are not only based on functions of the functional viewpoint, but also on elements of additional viewpoints. That means that a sequential arrangement of the design within a viewpoint or a level is not sufficient. The use of a parallel process instead allows temporal overlapping of the design activities of the different disciplines involved. In addition, it must be possible to integrate solution elements while increasing the level of abstraction (bottom-up). This results in the process flow as shown in Fig. 11-9 (right).

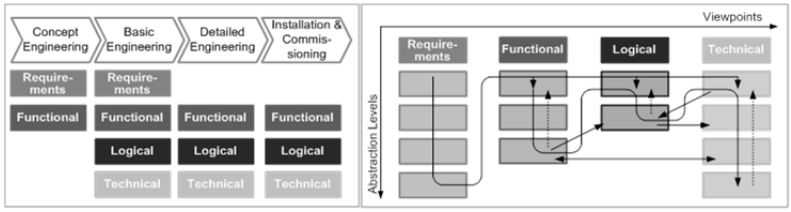


Fig. 11-9 *Mapping of the SPES modeling framework to the engineering process*

11.5 Summary

The goals of the automation domain were to optimize, interlink, and increasingly automate the engineering of automation systems using suitable models based on the SPES modeling framework. This requires integration of the discipline-specific models within the scope of structured, collaborative, and scalable engineering processes, also considering the physical and technical constraints of the automated system.

The concepts of the SPES modeling framework were mapped to the specific modeling languages of the automation domain. In this process, through intensive coordination with representatives from the automation domains and academic partners within SPES, it was found that the terms of automation engineering and those of the SPES methodology correspond well. Based on results from applying the SPES modeling framework in the context of several domain-specific case studies, we conclude that the automation engineering 4-tier metamodel of automation can be represented within the viewpoints and abstraction layers of the SPES modeling framework. Based on results from case studies, we were able to demonstrate that the SPES modeling framework can be applied in the automation domain.

Results also showed that the SPES modeling framework strengthens abstraction, hierarchy, and separation of concerns in interdisciplinary modeling and thus promotes interdisciplinary collaboration, which can be underlined from experiences from recent projects. Moreover, the emphasis on the functional and logical views supports automation engineering for shifting efforts from implementation into the early project phases, helping to reduce project risks as well as costs for the correction of errors and design decisions [Wagner et al. 2011]. However, there is currently no guideline for the application of the SPES modeling framework in the automation engineering process. Therefore, the SPES modeling framework has to be adapted further with regard to its

applicability in the automation domain. Specifically, this could be achieved through the integration of upcoming domain-specific challenges such as mechatronic design and design-by-reuse and by support through domain-specific application guidelines.

11.6 References

[Achatz and Löwen 2005] R. Achatz, U. Löwen: Industrieautomation. In: P. Liggesmeyer, D. Rombach (Eds): Software Engineering eingebetteter Systeme: Grundlagen, Methodik, Anwendungen. Munich: Elsevier, Spektrum Akademischer Verlag, 2005, pp. 497–525.

[Arendt and Taentzer 2012] T. Arendt, G. Taentzer: Model-driven engineering of composite model refactorings. In: Proceedings of the 34th International Conference on Software Engineering, 2012.

[Arendt et al. 2011] T. Arendt, S. Kranz, F. Mantz, N. Regnat, G. Taentzer: Towards syntactical model quality assurance in industrial software development: Process definition and tool support. In: R. Reussner, M. Grund, A. Oberweis, W. Tichy (Eds.): Software Engineering 2011: Fachtagung des GI-Fachbereichs Softwaretechnik, February 25-25, 2011 in Karlsruhe. Köllen Druck+Verlag, Bonn, 2011.

[Fay et al. 2011] A. Fay, T. Jäger, T. Wagner: Systematische Erfassung und Bewertung von gewerkeübergreifenden Schnittstellen in Engineering-Workflows. In: Proceedings des 8. Symposiums Informationstechnologien für Entwicklung und Produktion in der Verfahrenstechnik, Frankfurt am Main, Germany, 2011.

[Fieber et al. 2009] F. Fieber, N. Regnat, B. Rumpe: Assessing usability of model driven development in industrial projects. In: T. Bailey, R. Vogel, J. Mansell (Eds.): 4th European Workshop on "From code centric to model centric software engineering: Practices, Implications and ROI" (C2M). University of Twente, Enschede, 2009.

[Jäger et al. 2011] T. Jäger, A. Fay, T. Wagner, U. Löwen: Mining technical dependencies throughout engineering process knowledge. In: Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'2011, September 5-9, 2011, pp. 1-7.

[Lüder et al. 2010] A. Lüder, L. Hundt, M. Foehr, T. Wagner, J. Zaddach: Manufacturing system engineering with mechatronical units. In: Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'2010 September 13-16, 2010, pp. 1-8.

[Strube et al. 2010] M. Strube, A. Fay, S. Truchat, H. Figalist: Funktionale Anlagenbeschreibung als Basis der Modernisierungsplanung. In: VDI-Bericht Nr. 2092 Tagungsband VDI-Kongress AUTOMATION 2010, Baden-Baden, Germany, 2010.

[Strube et al. 2011] M. Strube, A. Fay, S. Truchat, H. Figalist: Durchgängige Modellunterstützung bei der Modernisierung komplexer Automatisierungssysteme. In: VDI-Bericht 2143 Tagungsband Automation 2011, Baden-Baden, Germany, 2011.

[VDI/VDE 2657] Verein Deutscher Ingenieure: VDI/VDE Guideline 2657 - Middleware in industrial automation (draft for guideline), ICS 35.240.50, 2010.

[Wagner and Löwen 2010] T. Wagner, U. Löwen. Modellierung: Grundlage für integriertes Engineering. In: VDI-Berichte Nr. 2092 Tagungsband VDI-Kongress AUTOMATION 2010, Baden-Baden, Germany, 2010.

[Wagner et al. 2011] T. Wagner, T. Jäger, A. Fay, H. Figalist: Systematische Risikominimierung im Engineering mit Abhängigkeitsanalyse und Schlüsseldokumenten – Vorgehen und Ergebnisse einer Fallstudie zur Erfassung der gewerkeübergreifenden Informationsschnittmenge im Engineering automatisierter Anlagen. In: VDI-Bericht 2143 Tagungsband Automation 2011, Baden-Baden, Germany, 2011.

[Wehrstedt et al. 2011]J.C. Wehrstedt, R. Rosen, A. Pirsing, C. Dietz: Simulation Based Engineering – Frühzeitige Validierung von Anlagenkonzepten. In: J. Gausemeier, F. Rammig, W. Schäfer, A. Trächtler (ed.): 8. Paderborner Workshop Entwurf technischer Systeme. HNI-Verlagsschriftenreihe Band 294, Paderborn, Germany, 2011.

Markus Fockel
Peter Heidl
Jens Höfflinger
Harald Hönninger
Jörg Holtmann
Dr. Wilfried Horn
Dr. Jan Meyer
Dr. Matthias Meyer
Jörg Schäuffele

12

Application and Evaluation in the Automotive Domain

This chapter summarizes the application and evaluation of the SPES engineering methodology in the automotive domain. After introducing the particular domain characteristics, we state some research questions that we have investigated. Some of the activities that address these research questions are presented in detail. We conclude that the SPES engineering methodology is a good basis for the development of automotive systems, but could be further refined to fit the particular needs of the domain.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_12, © Springer-Verlag Berlin Heidelberg 2012

12.1 Overview: Application Domain Automotive

With total revenue of €315 billion, and thus representing approximately 20% of German industrial production, the automotive industry is Germany's most important economic sector, offering employment to 714,000 people. Investing €20 billion in research and development, the automotive industry is the most innovative sector and thus contributes significantly to securing Germany as a location for business (source: VDA annual report 2011).

The necessary innovations are made possible by the enabling technology of embedded software systems. With regard to the total development effort, embedded systems make up between 30 and 40 percent of the value chain with an upward trend.

Across all domains, engineering of embedded systems is characterized by a physical context with real-time requirements and the necessity for interdisciplinary cooperation. Additionally, the automotive domain has a high share of quality requirements, cost pressure, and resource constraints. This is due to high product volumes ranging in the millions, particularly demanding safety and reliability requirements, and extensive variability stemming from a large number of system approaches and functional configurations.

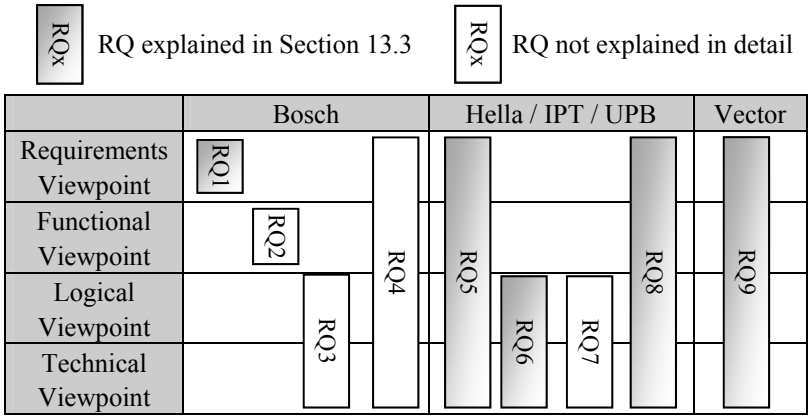
An integrated modeling approach is a key factor to mastering diverse requirements and being able to handle variants in highly complex surroundings. Both of these capabilities are vital in order to meet the challenges of the industry, which is why we worked on developing them in the SPES 2020 project.

12.2 Evaluation Strategy and Correlations to the SPES Modeling Framework

In the automotive application domain, the companies Hella KGaA Hueck & Co., Robert Bosch GmbH, and Vector Informatik GmbH, as well as the academic partners Fraunhofer IPT and University of Paderborn (UPB), worked on a variety of domain-specific *research questions* (RQs). To address the wide range of these questions, various research activities were conducted considering different types of case studies. RQs 1-4 focused on an evaluation of some of the approaches introduced in Part II of this book. RQs 5-8 evaluated and refined the SPES modeling framework (cf. Chapter 3) where necessary in order to exploit domain-specific information and to support domain-specific languages and

standards. RQ9 considered variant handling including tool support covering all viewpoints. The research activities conducted can be related to the viewpoints of the SPES modeling framework as shown in Tab. 12-1.

Tab. 12-1 *Overview of research questions in the automotive domain*



In the following, we will outline each RQ.

- *RQ1:* How can we apply model-based requirements engineering to the automotive domain? Employing the model types of the requirements viewpoint (see Chapter 4), we conducted case studies on an example engine control system and air system. Valuable input for the method developers was gathered from domain experts (see Section 12.3.1).
- *RQ2:* How can we address model-based function development throughout the automotive development life cycle? We analyzed the tool AUTOFOCUS3 by means of a case study conducted on an example engine control system.
- *RQ3:* How can we address safety design in model-based automotive development? The work on this RQ led to the development of the Vertical-Safety-Interface approach [Zimmer et al. 2011]. Industrial feedback was gathered via a case study conducted on a reallocation scenario.
- *RQ4:* How can we empirically validate the methods developed in the automotive domain? The resulting effect on complexity and efficiency was validated empirically in cooperation with Fraunhofer IESE.
- *RQ5:* How can we get from mainly informal requirements to an implementation based on the domain-specific AUTOSAR standard

[AutomotiveSIG 2010] in a more systematic way? A big challenge here was the assurance of consistency and traceability between artifacts of different development phases and viewpoints. A seamless model-based development methodology compliant to Automotive SPICE [AutomotiveSIG 2010] was developed. It addresses this research question using semi-automatic transitions between development phases and viewpoints [Holtmann et al. 2011b].

- *RQ6:* How can we identify design flaws regarding functional correctness and timing in early development phases? To address this question, different kinds of simulation techniques were integrated into the development methodology (cf. RQ5).
- *RQ7:* How can we apply the analysis techniques based on the SPES modeling framework (see Chapter 2) to AUTOSAR architectures? In order to answer this question, a concept for transforming AUTOSAR models into models corresponding to the SPES modeling framework was developed.
- *RQ8:* Is the conceived automotive development methodology (cf. RQ5) applicable and what are its limits? The methodology was evaluated by a proof of concept and developers were asked for feedback as to its feasibility.
- *RQ9:* How can we develop embedded systems for vehicles in a product line approach? Work on this RQ extended the concepts and features of the PREEvision tool to manage product lines and to derive consistent product variants of automotive embedded systems.

Some of these RQs are explained in more detail in the following section.

12.3 Detailed Experience Reports

In total, the RQs we addressed in our activities in the automotive domain cover all constructive development phases for automotive systems. In this section, we present a selection of our activities in more detail. Following the development process, we start with requirements engineering by presenting an evaluation of the requirements view (see Chapter 4) of an engine control system (see Section 12.3.1). We then continue (in Section 12.3.2) with a seamless development methodology from the requirements to an implementation based on AUTOSAR, which was evaluated on a body control module. This ECU offers a wide range of functions related to the car body (e.g., indicator control and interior light control) and communicates with several other ECUs. Finally, we present an approach for handling variants, which is necessary in all steps of the development process (Section 12.3.3).

12.3.1 Requirements Engineering Using the Model of the Requirements Viewpoint

In order to address RQ1 regarding the use of model-based requirements engineering in the automotive domain, we analyzed methods by applying them to the development of software for engine control systems. We analyzed this situation using an air system (system controlling the airflow to the engine) as an example.

Measuring the performance of model-based development methods, including model-based requirements engineering, relies heavily on how adequately models can describe the relevant characteristics of a system regarding both functionality and quality. Customer requests, testability of design and implementation artifacts, and shaping the development process are some of the key reasons for requirements engineering. The evaluation of the model of the requirements viewpoint was motivated by the following questions:

- Does the approach cover relevant functional problem classes?
- Does the approach support the step from the problem domain to the solution domain?
- Does the approach scale to large systems?
- Can the approach address the variability inherent to the domain adequately?
- Does the approach allow statements regarding the completeness or unambiguousness of requirements modeled?

Generally speaking, requirements in the automotive domain are documented in specification documents containing an informal textual description (cf. [Sikora et al. 2012]). This description is then supplemented by formal behavioral models such as MATLAB or ASCET for individual aspects. As described in Part II of this book, the requirements viewpoint (see Chapter 4) provides modeling concepts and language elements for:

- Static descriptions of requirements using context diagrams
- Dynamic descriptions of requirements using scenarios
- Traversal to the solution domain using function models

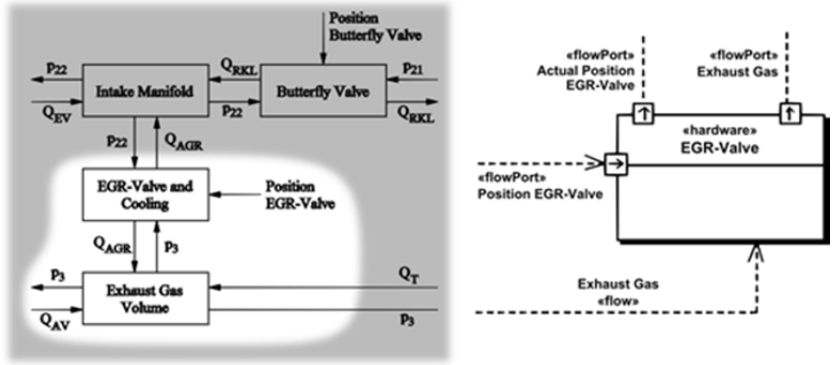


Fig. 12-1 *Comparing the plant structure (left) and the requirements viewpoint (right)*

The requirements viewpoint was evaluated in the context of the case study *Engine control system*. The aim of the evaluation was to identify which relevant domain characteristics are adequately supported and whether or not the efficiency and quality of development can be improved. For this purpose, context elements were compared to their modeling counterparts in the model of the requirements viewpoint and were evaluated with regard to the desired goals.

Does the approach cover relevant problem classes?

As a typical subsystem in an engine control system, the air system is characterized first and foremost by the system that is to be controlled. The context of the air system can be described from both a static perspective as well as a dynamic behavioral perspective using the plant, the user, the environment, and the control system. The individual context elements are characterized as follows:

- The context element *plant* represents the physics of the air system as a behavioral model in the form of differential equations. In order to specify requirements for the control of the plant, the target parameter we are striving to control (in this case the amount of air provided to the engine for combustion) must be put in relation over time along with other conditions (in this case, for example, outside air pressure). Additionally, the behavioral model that is synthesized from the physical equations of the system's components must be available. On closer inspection, the air system's component topology varies greatly between different operation phases, and thus the behavioral model varies as well. As a result, the operating air system presents itself as a multitude of variants that must respectively fulfill differing requirements in minute detail. The change between topologies occurs either due to external events or constellations within the system in

order to achieve required target values. In order to be able to control the target values, we need to know which information (sensor values) we have at our disposal and which physical parameters (actuators) we can influence. In our example, these are the throttle and the exhaust gas recirculation valve.

- The *usage* describes the intent with which the air system shall be operated and specifies the desired air amount that the air system control shall achieve. For example, the air system is operated using a specific pre-defined mix of fresh air and recirculated exhaust gas when regenerating the exhaust filter. Target value generation is continuous but may involve jumps when the operating situation changes spontaneously.
- The *environment* specifies information such as temperature, outside air pressure, battery voltage, and other environmental information relevant to the operation of the air system. These values are typically continuous.
- The *control system* involves describing information such as starting, operation, shutdown, or test modes as discrete events.

Evaluation of the static perspective

As described above, all value- and event-discrete elements can be adequately and completely described using a context diagram. Time- and value-continuous elements can be approximated by listing the main aspects of the continuous contextual element.

Evaluation of the dynamic perspective

Interactions between system and system context can be described very well using scenario diagrams. The question of which intents are used to operate the system is addressed particularly well. As far as continuous interactions between system and plant are concerned, this behavior must first be transferred to a discrete form. This only works well if the plant's behavior can be assumed to be continuous. While applicable for a wide range of operating situations of the air system, it cannot be assumed for all physical plants.

Addressing the problem classes in the context model

In summary, the question of an adequate problem description using the contextual view with the modeling concepts of the requirements viewpoint is depicted in Tab. 12-2. Value- and event-discrete behavior is addressed fully, whilst value- and time-continuous behavior can only be approximated through the use of simplifying discretization and thus remains incomplete.

Does the approach support the traversal from the problem domain to the solution domain?

Using value- and time-discrete semantics, behavior can be easily documented in an understandable form. For embedded systems, it is of utmost importance that they remain operational in all situations and thus expectations regarding completeness are extremely high. A scenario-based approach, however, is seldom complete or free from ambiguities. The value- and event-discrete semantics can only approximate value- and

time-continuous behavior and thus remain imprecise and, for our purposes, incomplete. The traversal to the solution domain relies on an approach based on value- and event-discrete sequences, ultimately leading to a state machine. If, however, as in our case, value- and time-continuous behavior is dominant, the solution must be based on a cybernetic control system approach utilizing signal-theoretic thinking.

Tab. 12-2 *Coverage of problem classes using the requirements viewpoint model*

Contextual Element	Problem Class	Requirements Viewpoint Model Type	Covered?
Plant	Value- and time-continuous	Context diagram	Partly
Environment	Value-discrete and -continuous	Context diagram Scenario diagram	Widely
Usage	Value-discrete Event-discrete	Context diagram Scenario diagram	Fully
System Control	Event-discrete	Scenario diagram	Fully

In order to be applicable to the domain of physically dominated systems, the requirements viewpoint model must be developed further in this direction. Under this premise, software can be developed based on system requirements resulting from systems co-design of all system disciplines concerned.

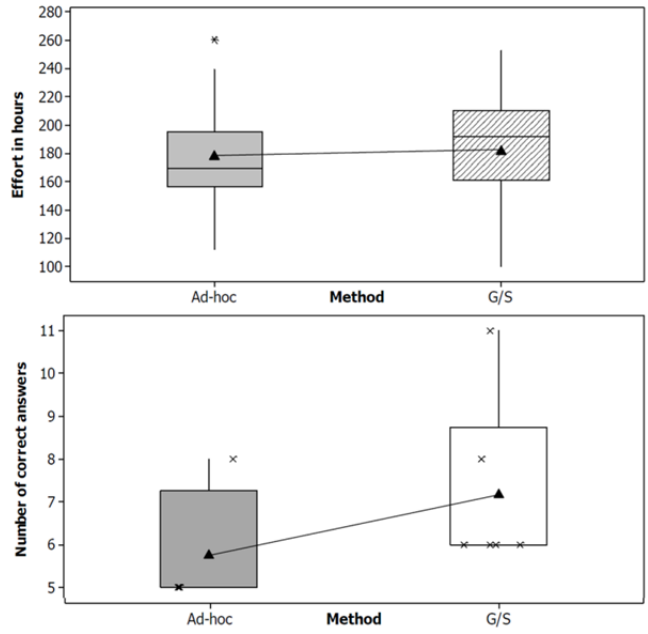


Fig. 12-2 *Effect on effort (top) and system comprehension (bottom)*

The following conclusions were reached by means of a case study [Gross et al. 2009] that was performed to assess the effect of the approach on efficiency and quality in the automotive domain (see Fig. 12-2). Firstly, the approach is neutral with regards to effort and quality (see Fig. 12-2, top). Secondly, the distribution of effort is shifted towards earlier development phases. Thirdly, communications between stakeholders and system understanding among developers were improved (see Fig. 12-2, right).

12.3.2 A Seamless Development Methodology for Automotive Systems

As seen in the evaluation of RQ1, the SPES engineering methodology has to be tailored for specific domains. In order to address RQ5, the SPES modeling framework was refined for the application within the automotive domain. We evaluated the concepts by means of the case study *Body control module*.

Requirements and functional viewpoints

Restricted natural language for requirements specification

As already stated in the last section, today, requirements are mostly specified in unrestricted natural language [Sikora et al. 2012]. The informal character of natural language, as well as its inherent ambiguity, can lead to inconsistent, ambiguous, and incomplete requirements. To resolve this problem, we use *requirement patterns* that are textual templates for different types of requirements [Kapeller and Krause 2006, Holtmann 2010]. Requirement types supported that reside in the requirements viewpoint (see Chapter 4) are, among other things, solution-oriented (see Section 4.2.4), timing, and safety requirements. However, functional requirement patterns describe the functional decomposition of the system under development (SUD) and reflect a functional hierarchy as explained in Chapter 5. Example 12-1 shows a functional requirement pattern in the upper part. The parts in square brackets are optional, and parts in angle brackets are variable and replaced by functionalities of the SUD. The lower part of Example 12-1 represents an instance of this requirement pattern and describes an excerpt of the functional decomposition of the SUD into its functionalities.

Example 12-1: Requirement pattern and instance

The functionality of the system "<system>" consists of the following function[s]: <function list>.

The functionality of the system "Control Indicators" consists of the following functions: Indicate, Switch Hazard Lights.

Processing of textual requirements for automated validation and transition to model-based design

Requirements formulated by means of requirement patterns are derived from the informal requirements systematically using a process refining the one used within the requirements viewpoint (see Section 4.4). Since we restricted the expressiveness of natural language and hence disambiguated it, the resulting requirements are understandable to all stakeholders and can be automatically processed at the same time. Thus, they can be validated automatically to ensure consistency or completeness, for example (see Fig. 12-3, left) [Holtmann et al. 2011a]. Furthermore, to ease the transition to model-based development, they are automatically transformed into a system analysis model using a Triple Graph Grammar (TGG) [Schürr 1995]. TGGs specify rules for bidirectional model-to-model transformations (M2M) that can be executed automatically and also preserve traceability and consistency. Requirements are formulated according to the requirement patterns and thus, the analysis model is kept traceable and consistent. By enabling the automatisms mentioned above, this procedure goes beyond the manual modeling of the SUD functions according to informal requirements as explained in Chapter 5.

Logical viewpoint

In the logical viewpoint, the logical component architecture is designed, as explained in Chapter 6, on the basis of the system analysis model resulting from the functional viewpoint. The elements of the analysis model are allocated to elements of the logical component architecture to document which logical components realize which functions and to maintain traceability [Meyer et al. 2011], see Fig. 12-3. For example, the functions *Indicate* and *Switch hazard lights* are allocated to two logical components, *Indicator* and *BrakeLampActuator*, respectively. The semantic correctness of the allocations can be ensured with the mapping relations based on contract-based design introduced in Part II.

Transition to AUTOSAR application software

To support a seamless development process, the logical component architecture can be automatically transformed into AUTOSAR application components. To do this, elements to be transformed into AUTOSAR application software (ASW) are marked manually using a

profile. The resulting refined logical component architecture is then automatically transformed using model-to-model transformations with TGGs (see Fig. 12-4) [Giese et al. 2010]. The usage of TGGs again allows for preservation of traceability and consistency between the models.

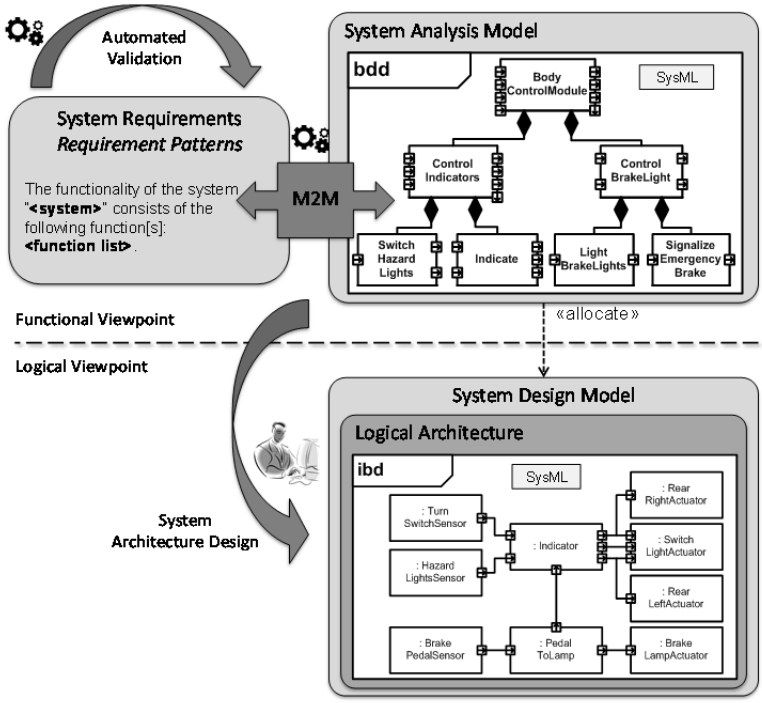


Fig. 12-3 *From the functional to the logical viewpoint*

Functional simulation of AUTOSAR application software

After the transformed AUTOSAR ASW has been manually enriched with behavior, this behavior can be simulated using the COTS tool SystemDesk[2] to validate its functionality before the software is deployed on hardware, see Fig. 12-4, upper right. Therefore, the compiled code of the overall SUD is executed, and thus its responses with regard to predefined stimuli are generated. The dynamic system behavior of the

[2] http://www.dspace.com/systemdesk/

interconnected software components can thus be tested at an early design stage, which addresses RQ6.

Technical viewpoint

Resource and real-time simulation

We further enrich the system design model with information from the technical viewpoint (see Chapter 7), as seen in Fig. 12-4, lower left. For example, a task calling operations of the logical component *Indicator*, its activation policy, and its allocation to an executing CPU are specified.

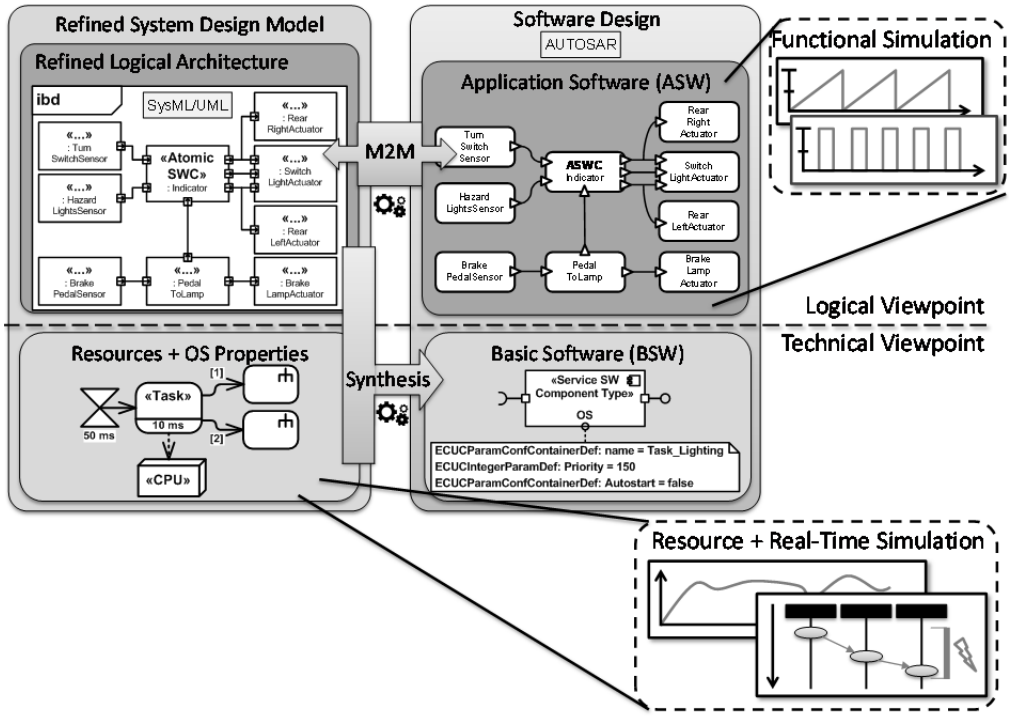


Fig. 12-4 *From the logical to the technical viewpoint*

On the one hand, we use this refined system design model to integrate a resource and real-time simulation of the SUD (see Fig. 12-4, lower right) to support architectural decisions and to validate the architecture

concerning timing requirements. This was realized by implementing a generator for simulation models for the COTS tool chronSIM[3] [Nickel et al. 2010], which allows computation of the CPU loads and simulation of the timing behavior of the SUD. Furthermore, timing requirements can be formalized by means of requirement patterns (see above) and used within the simulation to directly indicate requirement violations within the simulation [Meyer et al. 2011]. The simulation of the scheduling for a specific execution platform enables identification of design flaws in early development phases before first prototypes are available, addressing RQ6.

Partial synthesis of AUTOSAR basic software configurations

On the other hand, we use the refined system design model to further simplify the transition to AUTOSAR. In addition to the ASW, an AUTOSAR model consists of basic software (BSW) and an automatically generated middleware. Typically, for most parts the BSW is configured and then its source code is generated automatically. However, since there are thousands of different possible configurations depending on the architecture, support must be provided for this configuration task. In order to (semi-)automate the configuration, we developed an algorithm (see Fig. 12-4) for synthesizing parts of the AUTOSAR configuration [Meyer and Schäfer 2009]. For example, we preconfigure the operating system and the communication stack [Meyer and Holtmann 2011]. Thus, the main architecture decisions that were already specified within the refined system design model are transferred to the AUTOSAR basic software configurations.

Evaluation

In order to answer RQ8, the development methodology presented for automotive systems was evaluated with an excerpt of the case study *Body control module* by a proof of concept and structured expert interviews. For the evaluation, three experts from Hella and another industrial partner from the automotive sector were interviewed after using this new approach within a prototype. Of course, the number of persons interviewed is too small to gain a universally valid result, but we tried to minimize this effect by choosing experts for every development phase.

[3] http://www.inchron.com/chronsim.html

Furthermore, some concepts have already been applied in real development projects.

Consistency and traceability is maintained throughout the development process

The proof of concept demonstrated that the systematic and partially automated transitions between the different development phases and viewpoints, together with automated checks, ensure traceability and consistency throughout the methodology, especially if parts of an artifact in a development phase are added or removed. For example, newly added requirements result in new functions within the analysis model. In this case, the fact that there is no logical component that takes care of this function is revealed automatically. Furthermore, the expert interviews demonstrated that manual and thus extensive developer tasks are reduced by the transitions, and that the integration of simulation techniques in early development phases reduces extensive iteration loops.

12.3.3 Variant Handling

Embedded electronic systems are designed to fulfill not only the requirements of a single vehicle, but also the requirements of an entire family of vehicles. Therefore, as described in RQ9, the development methodology must address variant management in order to be applicable for the automotive domain. The information model for designing automotive embedded systems in the PREEvision tool is aligned to the SPES modeling framework and supports the systems engineering principles of abstraction, modularization, and reuse.

Information model for designing automotive embedded systems

Every SPES viewpoint can be assigned to one or several PREEvision modeling layers. In PREEvision, product lines of electronic vehicle systems are modeled from requirements to the hardware geometry using seven main layers.

SPES viewpoints and PREEvision modeling layers

On the top level, the design starts with the definition of requirements and customer features. These layers correspond to the SPES requirements viewpoint. Vehicle features can be defined in PREEvision using requirements. Customer features describe a set of features of the vehicle from the vehicle user's perspective. The customer features are organized in a tree representing a superset of features to be fulfilled by a vehicle product line. This is often referred to as a "150% model."

Example 12-2: Engine variants

All engine variants are considered in the product line, but only one engine variant will be delivered and finally built in a vehicle. All transmission variants are considered, but only one transmission will be selected later on. All optional features are considered, but only a subset will be chosen later on.

In practice, a two-step-approach for variant management is often applied — from a vehicle product line ("150% model") to a vehicle family (often also called "120% model") to a concrete vehicle ("100% model"). Due to the high number of options, it is not possible to manage each and every variant explicitly. Therefore, typically 150% models and 120% models are designed.

PREEvision supports the definition of variant conditions between features. Such variant conditions can be, for example, "exclusive-or" relations or "needs" relations between features, but also "or" and "optional" relations.

The result is a customer feature model (cf. Fig. 12-5) similar to the FODA approach [Kang et al. 1990]. In production projects, the customer feature tree is typically organized according to the vehicle manufacturer's organizational units following the classical division powertrain, chassis, body, comfort, and multimedia.

Fig. 12-5 *Customer feature model in PREEvision*

The next design level is the functional viewpoint, referred to as the logical architecture in PREEvision. Hierarchies are used to structure the complete layer according to the organizational responsibilities at the

vehicle manufacturer. However, logical signal connections crossing the organizational responsibilities can also be modeled by system diagrams — or activity chains can be used to express a logical control sequence from the sensors to the actuators realizing a given customer feature. This means that customer features are represented by an orthogonal subset of the organizational structure of the logical architecture.

The "realization" or "implementation" relations between customer features and logical architecture are specified using mappings. Mappings are information objects that decouple all the modeling layers. This enables the design engineer to handle even variants of mappings.

Example 12-3: Mapping variants between logical architecture and the hardware architecture

In practice, the logical architecture and the hardware architecture are often mapped to each other in a variant-specific manner. In this case, only the mappings are variant-specific — the logical and the hardware architecture are neutral.

All subsequent modeling layers in PREEvision are used to specify the logical and the technical viewpoints in the SPES modeling framework. Here, PREEvision differentiates between system software architecture to model the logical viewpoint and the software implementation and hardware architecture to model the technical viewpoint.

The system software architecture supports the AUTOSAR platform [AUTOSAR 2011], including the implementation of software components. The hardware architecture is modeled in several abstractions representing the hardware network topology, the hardware components, the schematics, and wiring harness. Geometry data are stored in the technical viewpoint.

The communication layer supports the definition of conventional and bus signals, protocol data units, frames, and communication schedules.

Variant management

A product variant is defined in PREEvision as a consistent subset of a product line over all layers. Challenges from a tool perspective are the consistency and completeness of the defined subsets: they have to be guaranteed inside every layer, but also across the layers.

Questions such as "Are all input signals needed provided in a given variant?" or "Are all software components that are part of a given variant mapped to hardware components?" have to be answered with "yes." This can be checked by user-defined consistency checks and propagation rules

that can be designed on a customer-specific basis — and that guarantee consistent and complete variant models.

The approach also has to be usable by an organization following a defined process with distributed responsibilities and defined roles. Examples for roles are persons responsible for software, persons responsible for hardware, and system architects. Database functions are available in PREEvision to prevent concurrent access to an artifact, but also conveniently support concurrent work on different artifacts, history, and archive functions.

Variant design is supported by engineering features such as the signal router taking into account only a selected variant and not always the complete product line. Highlighting of variants is supported in all diagrams, and metrics are available to calculate characteristics of a product variant such as bus loads, weight, or costs.

We succeeded in developing concepts for the integration of Simulink as an implementation tool for software components. Furthermore, usability, convenience, and adaptation functions were designed for the usage of PREEvision in production projects. Extended feature modeling capabilities were developed and the support of distributed responsibilities and database functions was optimized. Further research topics included consistency analysis not only inside a given layer, but also across layers.

12.4 Summary

Several challenges arise due to characteristics inherent to the automotive domain. Although all aspects of development are addressed by processes employed, significant gaps are often present between process steps. This also applies to the artifacts these steps respectively produce or use, further affecting traceability between them. Requirements are usually documented in informal natural language, introducing the possibility of ambiguity, inconsistency, and incompleteness, and must be considered by formal models used in later design phases. Embedded systems must cope with continuous plant behavior that is both difficult to specify as well as validate, and even more difficult to control, using systems tailored toward discrete behavior. Furthermore, embedded systems in the automotive domain are becoming more complex and it is therefore difficult to validate in early design phases whether the specified architecture is correct. Additionally, systems in the automotive domain are usually designed for entire product lines, making variant management important for handling the resulting complexity of development.

Many of these aspects have now been addressed and at least partly solved by means of the methodologies developed to answer the research questions presented in Section 12.2. As a whole, the SPES methodology is a very good basis, but also leaves room for further development for refining and adapting the approaches for specific application domains.

12.5 References

[AutomotiveSIG 2010] Automotive Special Interest Group (SIG): Automotive SPICE. Process Reference Model. Accessed on: April 3, 2012. http://www.automotivespice.com/automotiveSIG_PRM_v45.pdf.

[AUTOSAR 2011] AUTOSAR GbR: Specification of ECU Configuration. http://www.autosar.org/download/AUTOSAR_ECU_Configuration.pdf. Accessed on April 3, 2012.

[Giese et al. 2010] H. Giese, S. Hildebrandt, S. Neumann: Model synchronization at work: Keeping SysML and AUTOSAR models consistent. In: G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, B. Westfechtel (Eds.): Graph Transformations and Model-Driven Engineering. Springer, Berlin/Heidelberg, 2010; pp. 555–579.

[Gross et al. 2009] A. Gross, J. Dörr, I. Menzel, M. Müller: Use Cases vs. Funktionale Spezifikation: Ein experimenteller Vergleich zweier Techniken zur Anforderungsspezifikation, GI-Fachgruppen-Treffen Requirements Engineering, 2009.

[Holtmann 2010] J. Holtmann: Mit Satzmustern von textuellen Anforderungen zu Modellen. In: OBJEKTspektrum, Vol. RE/2010, 2010, http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2010/RE/holtmann_OS_RE_2010.pdf. Accessed on: March 31, 2012.

[Holtmann et al. 2011a] J. Holtmann, J. Meyer, M. von Detten: Automatic validation and correction of formalized, textual requirements. In: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2011. IEEE Computer Society, Los Alamitos, 2011, pp. 486–495.

[Holtmann et al. 2011b] J. Holtmann, J. Meyer, M. Meyer: A seamless model-based development process for automotive systems. In: R. Reussner, A. Pretschner, S. Jähnichen (Eds.): Software Engineering 2011 – Workshopband (inkl. Doktorandensymposium). Bonner Köllen Verlag, Bonn, 2011, pp. 79–88.

[Kang et al. 1990] DTIC: Feature-Oriented Domain Analysis (FODA) Feasibility Study. http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA235785. Accessed on April 3, 2012.

[Kapeller and Krause 2006] R. Kapeller, S. Krause: So natürlich wie Sprechen – Embedded Systeme modellieren. In: Design & Elektronik, 2006/08; pp. 64–67.

[Meyer and Holtmann 2011] J. Meyer, J. Holtmann: Eine durchgängige Entwicklungsmethode von der Systemarchitektur bis zur Softwarearchitektur mit AUTOSAR. In: H. Giese, M. Huhn, J. Philipps, B. Schätz (Eds.): Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII. fortiss, Munich, 2011, pp. 21–30.

[Meyer and Schäfer 2009] J. Meyer, W. Schäfer: Automatische Analyse und Generierung von AUTOSAR-Konfigurationsdaten. In: H. Giese, M. Huhn, U. Nickel, Bernhard

Schätz (Eds.): Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme V. Carl-Friedrich-Gauß-Fakultät für Mathematik und Informatik, Technische Universität Braunschweig, 2009, pp. 82–91.

[Meyer et al. 2011] J. Meyer, J. Holtmann, M. Meyer: Formalisierung von Anforderungen und Betriebssystemeigenschaften zur frühzeitigen Simulation von eingebetteten, automobilen Systemen. In: J. Gausemeier, F. Rammig, W. Schäfer, A. Trächtler (Eds.): 8. Paderborner Workshop Entwurf mechatronischer Systeme. Heinz Nixdorf Institut, Paderborn, 2011, pp. 203–215.

[Nickel et al. 2010] U. Nickel, J. Meyer; T. Kramer: Wie hoch ist die Performance?. In: Automobil-Elektronik, Vol. 2010, No. 3, 2010, pp. 36–38.

[Schürr 1995] A. Schürr: Specification of graph translators with triple graph grammars. In: E. W. Mayr (Eds.): Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science, Vol. 903, Springer, Berlin/Heidelberg, 1995, pp. 151–163.

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal, Vol. 17, No.1, 2012, pp. 57-78.

[Zimmer et al. 2011] B. Zimmer, S. Bürklen, M. Knoop, J. Höfflinger, M. Trapp: Vertical safety interfaces - improving the efficiency of modular certification. In: Proceedings of the 30th International Conference of Computer Safety, Reliability, and Security, 2011.

Ottmar Bender
Martin Hiller
Dr. Maurice Girod
Carsten Strobel
Martin Waßmuth
Laurent Dieudonné

13

Application and Evaluation in the Avionics Domain

The SPES 2020 partners Airbus, Cassidian, EADS Innovation Works, and Liebherr from the avionics domain formed the avionics group in the SPES 2020 research program. This group worked together on the SPES 2020 challenges, i.e., modeling of heterogeneous embedded systems, requirements, platform architectures, safety, certification, and multicore architectures. Through SPES 2020, a significant improvement in development methods for requirements engineering, model-based systems engineering, model-based software engineering, and verification has been achieved in the avionics domain.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_13,

13.1 Overview: Application Domain Avionics

Avionics systems have to be certified by the airworthiness authorities before they can be installed and operated in an aircraft. In order to achieve certification for an avionics system, the company developing and manufacturing this system has to provide the airworthiness authorities with the evidence that the system is safe.

Before an embedded system can be operated in an aircraft, the developer has to provide evidence to the airworthiness authorities that the probability of hazards is below predefined risk levels. In addition, the system itself, its documentation, as well as the development process must follow the standards defined in the planning phase strictly.

Consequently, the methods developed in SPES 2020 must support and comply with the safety analysis and development processes in the avionics domain and allow the engineering of safe systems, including hardware and software. In this chapter, we explain how the SPES modeling framework was applied to the development processes in the avionics domain and evaluated with regard to the fulfillment of the domain-specific requirements outlined in Section 2.2.3.

13.2 Evaluation Strategy and Application to the SPES Modeling Framework

This section describes the strategy that was pursued in order to evaluate the SPES modeling framework. Based on the initial situation in the avionics domain (cf. Section 2.1.3), and the peculiarities of this application domain (see Section 13.1) and its specific requirements, a case study was defined and used to assess to what extent the requirements for the SPES modeling framework (cf. Section 2.2.3) have been met. This case study is illustrated in Section 13.2.1. An overview of its application is given in Section 13.2.2, while Section 13.3 discusses the results from the evaluation.

13.2.1 Avionics Case Study

Problems considered in the avionics application domain

The avionics group defined an avionics case study to express selected pressing problems such as the formalization and structuring of requirements, bridging the gap between system and safety engineering, automated test case and procedure generation, and the seamless system and software development flow from requirements to design. The

avionics case study describes an example of the aircraft functions *situation awareness, flight control,* and *air conditioning*. The case study focuses on these functions to represent and show the problems listed above.

The *situation awareness* function presents the traffic situation around the aircraft to the flight crew. The situation is shown as pictures on display units. These pictures consist of information about other aircraft, flight corridors and areas, and flight-relevant geological formations. In SPES, this function was used to represent the problems of the seamless system and software development flow and bridging the gap between system and safety engineering.

The *flight control* is a fly-by-wire system that sends all *flight control* commands from the flight controls to the actuators (see Fig. 13-1) via a communication network. In SPES, this function was used to represent the problem of automated test case and procedure generation.

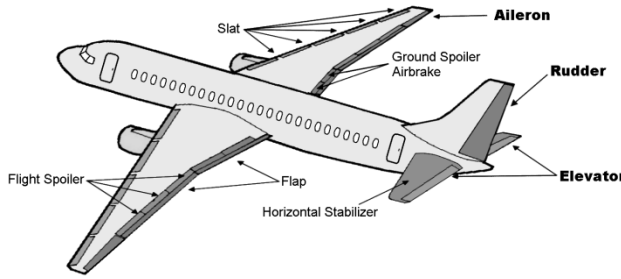


Fig. 13-1 *Flight control*

The *air conditioning* system enables the passengers to stay in the aircraft interior during the flight with no major restrictions or effects on health. In particular, it allows them to *breathe without equipment* and to *feel comfortable in indoor clothing*. In SPES, this function was used to represent the problem of the formalization and structuring of requirements.

13.2.2 Application of the SPES Modeling Framework

Structuring an aircraft

The following paragraphs show how the SPES modeling framework is applied to selected parts of the avionics case study.

Typically, an aircraft is structured into the following types of system elements: aircraft, system, equipment, hardware, and software. The term *system element* is used to address any structural element of an aircraft, e.g., equipment, hardware, or software. This type of structure is the

motivation for instantiating the generic abstraction layers of the SPES modeling framework (see Section 3.5) to the layers shown in Fig. 13-2. The generic SPES viewpoints (i.e., requirements viewpoint, functional viewpoint, logical viewpoint, and technical viewpoint) of the SPES modeling framework are used for system and software modeling in the avionics domain.

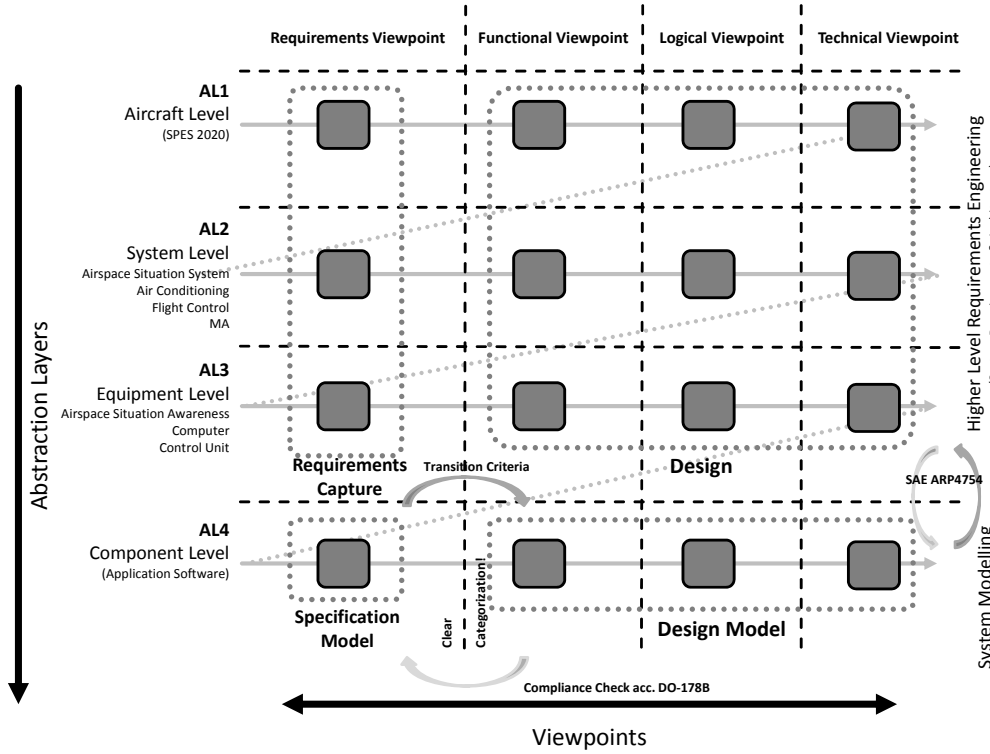


Fig. 13-2 *Abstraction layers and viewpoints in the avionics case study*

This concept of layers and viewpoints has to be mapped to the structure of the repository for the system and software modeling. The avionics group defined a repository structure to model the case study using SysML with profiles, e.g., component fault trees (CFT). The rules and principles derived from these concepts have been documented in the avionics modeling guide.

The avionics model repository complies with the abstraction layers and viewpoints philosophy as defined in SPES 2020. The layout of the avionics model is shown in Fig. 13-3. In this repository structure, the

layers are realized as SysML packages and the viewpoints are realized as packages within these layer packages.

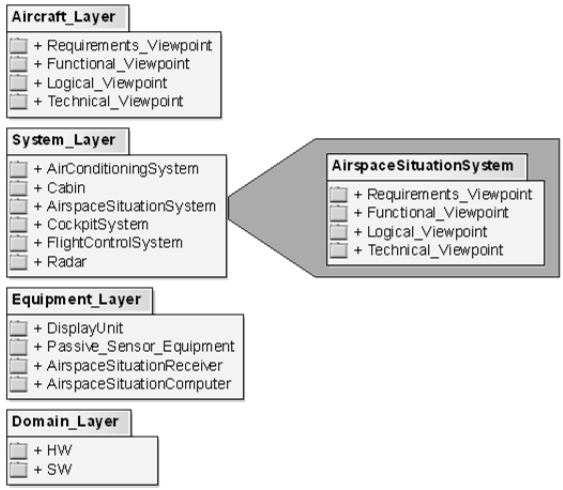


Fig. 13-3 *Repository structure*

13.3 Evaluation Results for each Viewpoint and Quality Aspect

This paragraph describes the viewpoints applied to the avionics domain with their purpose, modeling activities, and the resulting model elements. It is important to state that the activities described in the viewpoints may be performed iteratively or even in a partially ordered sequence. This means that the concept of viewpoints and abstraction layers does not prescribe a certain sequence of activities, e.g., left to right or top-down. Quality aspects can be added to a viewpoint to model cross-sectional subjects, e.g., safety. Where applicable, the quality aspects *safety* and *real-time* have been considered as part of the viewpoints.

13.3.1 Requirements Viewpoint in the Avionics Domain

Activities in the requirements viewpoint

The purpose of the requirements viewpoint is to show the goals, requirements, actors, use cases, interfaces, and scenarios and their interrelation. The avionics group specified the following activities to describe the system element defined by the scope of this viewpoint and the related abstraction layer:

- Elicit aircraft, system, equipment, hardware, and software requirements from stakeholder goals.
- Trace the specified requirements to requirements from the abstraction layer above. This connects the adjacent abstraction layers.
- Identify the actors and develop the use cases from the defined requirements.
- Trace the use cases to the requirements. This traces the use cases indirectly to the requirements of the abstraction layer above.
- Identify the interfaces from the relation between the actors and use cases.
- Develop scenarios from use cases. These scenarios are the basis for validating and verifying the requirements, and provide the basis for the functional architecture of the functional viewpoint.

Case study: Requirements

The avionics case study gives an example of the requirements viewpoint for the aircraft abstraction layer. The top level use case diagram for the *situation awareness* function is shown in Fig. 13-4. It describes the identified actors *crew, other aircraft,* and *airspace information system*. The interfaces of the aircraft function are derived from the communication links between these actors and the use case *Display airspace situation*. The use case originates from the requirements it traces to.

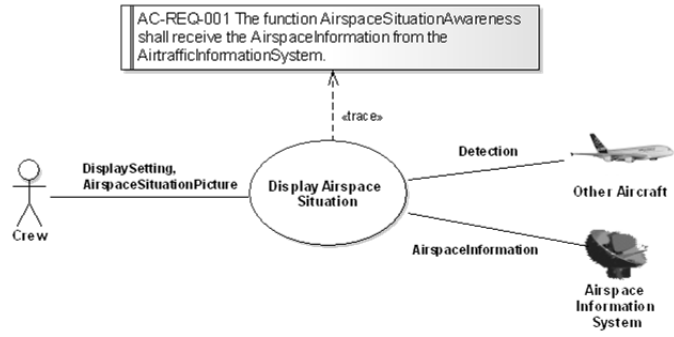


Fig. 13-4 *Top-level use case diagram in the requirements viewpoint*

One of the aircraft functions addressed by the avionics case study is the air conditioning. In this case study, the text-based specification was transferred into a model-based specification using the SPES modeling

framework (multiple abstraction layers and viewpoints) and a specific modeling tool.

Implementation of the SPES modeling framework

The SPES modeling framework is implemented as a plug-in to a modeling tool. This SPES 2020 plug-in offers certain predefined model elements, based on the "rich component" concept, that are the foundation for automated model analysis. In the modeling tool, there is a graphical model view as well as a model tree reflecting the model's element structure. The predefined model elements offer a predefined model tree structure whose elements, e.g., ports, are instantiated by the concrete case. The following paragraphs show a modeling example.

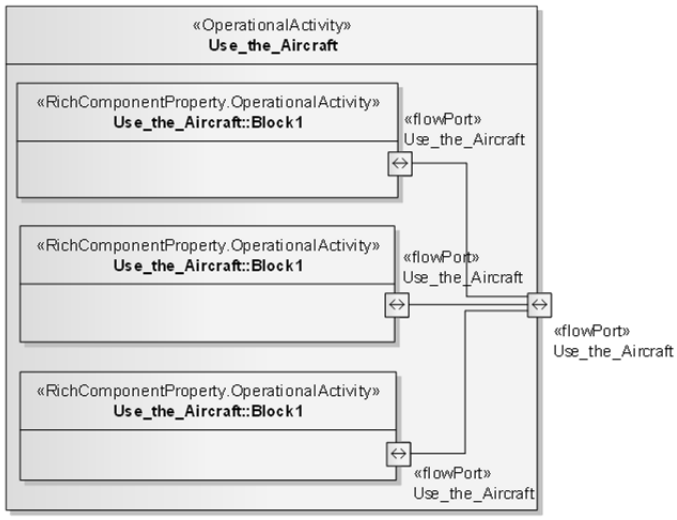


Fig. 13-5 *Requirements viewpoint in the aircraft layer*

The top abstraction layer is the aircraft layer. Its requirements viewpoint maintains the passengers' overall use case, i.e., use the aircraft for flying from origin to destination. For the aircraft, this means the passengers *board the aircraft*, *stay in the interior during flight*, and *leave the aircraft*. These activities are linked to the use case by port relations. The graphical model view is shown in Fig. 13-5.

The requirements viewpoint of the next lower abstraction layer, i.e., the system layer, focuses on the actual air conditioning system. This system enables the passengers to stay in the aircraft interior during the flight. In particular, it allows them to *breathe without equipment* (oxygen is provided) and to *feel comfortable in indoor clothing* (heating is provided), as graphically modeled in Fig. 13-6.

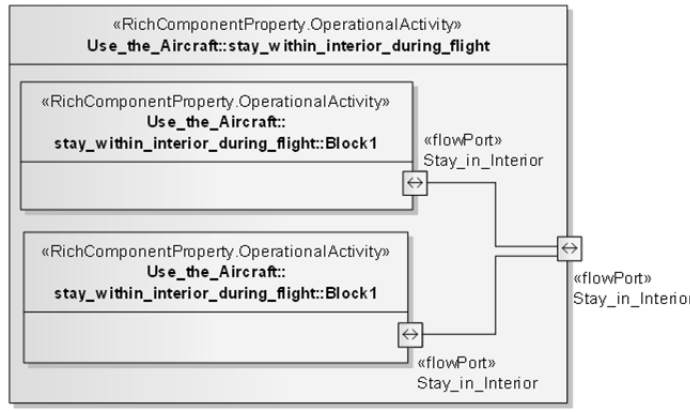


Fig. 13-6 *Requirements viewpoint in system layer*

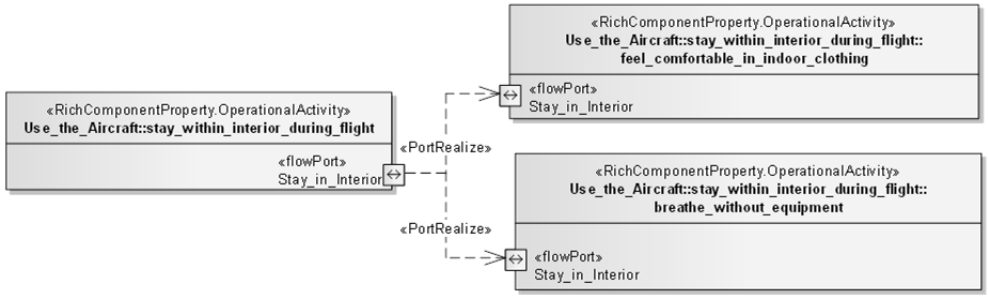


Fig. 13-7 *Port mapping in system layer and mapping*

The link between the requirements viewpoint of the aircraft layer and the requirements viewpoint of the system layer is modeled by a mapping diagram, based on port mappings as shown in Fig. 13-7.

As a conclusion, we can confirm that it is possible to transfer a text-based specification into a model-based representation using the SPES modeling framework.

13.3.2 Functional Viewpoint in the Avionics Domain

Activities in the functional viewpoint

The purpose of the functional viewpoint is to show the functions, subfunctions, and functional interfaces of the system element in question. The avionics group specified the following activities to describe the system element for this viewpoint:

- Derive aircraft, system, equipment, hardware, and software functions from scenarios. The scenarios are described in the requirements viewpoint. Remember that scenarios are linked to requirements and therefore the functions are indirectly linked to the requirements as well. In the avionics domain, it is mandatory to have full traceability between all of the requirements of the different layers. It is good practice to have traceability between the model elements of the viewpoints within a layer.
- Define the functional architecture by decomposing these functions into subfunctions and specifying their interfaces. This decomposition ends when each subfunction can be clearly mapped to a logical system element (see logical viewpoint, Section 13.3.3). This decomposition of functions into subfunctions typically creates new interfaces between them. The subfunctions identified on the lowest level constitute the functions of the next lower layer.
- Perform a functional hazard analysis (FHA) to identify and assess the risks of the system element.

Case study: Functional decomposition

Fig. 13-8 shows how the aircraft function *situation awareness* is linked to the use case *Display airspace situation* (see Fig. 13-4). This is an example of how the requirements viewpoint is linked to the functional viewpoint.

Fig. 13-9 shows the functional architecture of the *situation awareness* function within the functional viewpoint. This example shows the decomposition of the *situation awareness* function into its subfunctions and the newly defined interfaces *Track* and *AirspaceSymbol* between these subfunctions. The external functional interfaces are modeled as activity parameter nodes and the internal functional interfaces are modeled as action pins. The external interface has been defined in the requirements viewpoint in the use case modeling.

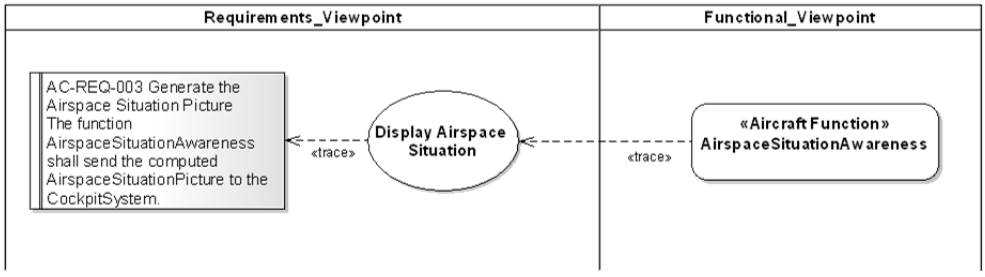


Fig. 13-8 *Transition from the requirements to the functional viewpoint*

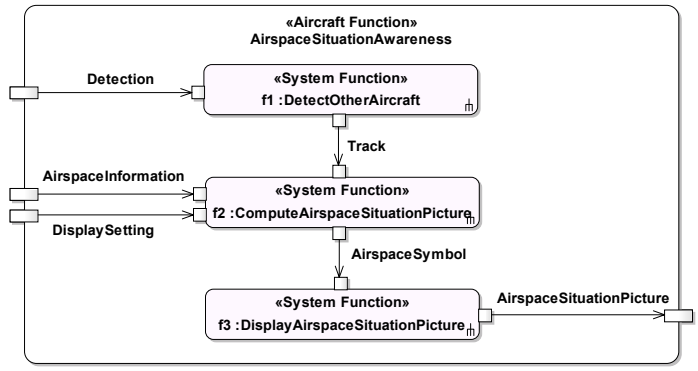


Fig. 13-9 *Functional viewpoint*

Case study: Safety cases

Traditional manual approaches are not applicable

This section shows an example of the safety aspect as part of the functional viewpoint. It shows the elaboration of safety cases within the SPES modeling framework. Due to the rising complexity of modern airborne systems, traditional manual safety assessment and certification workflows are becoming an increasing burden in aircraft development. According to a survey among aviation experts, safety and certification efforts together amount to more than 50 percent of the overall development effort. Safety analyses, proofs, and argumentations for certification purposes are mainly performed manually without coherent tool support. As a consequence, the production of comprehensive and reliable documentation of an aircraft's airworthiness in the form of safety cases is a lengthy and expensive manual process.

Concept for integrated safety cases

Model-based analysis

We introduce a safety case approach that takes the suggested safety assessment process introduced in [Waßmuth and Stilkerich 2011] into account. The SPES 2020 Safety Case concept favors the graphical argumentation of the system's safety based on the design model and the corresponding component fault trees introduced in [Domis and Trapp 2009]. In SPES 2020, the avionics group developed a concept for seamless and iterative safety assessment to support the certification process by deriving safety cases, see Fig. 13-10.

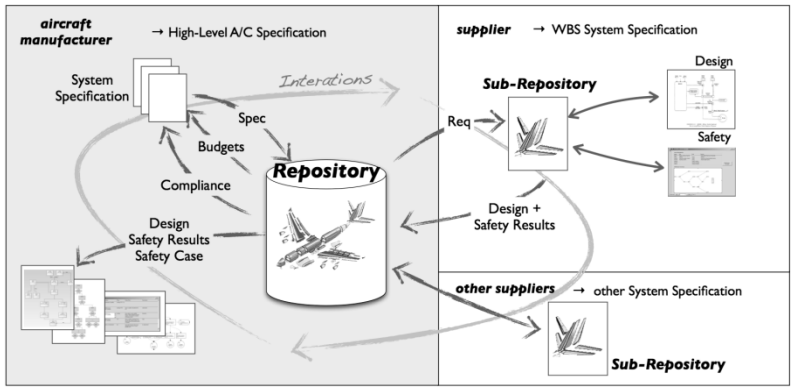


Fig. 13-10 *Safety assessment concept — repository for integrated and seamless systems engineering, safety assessment, and certification*

The centralized repository integrates all relevant design and safety information to perform the required safety analyses automatically. The information comprises requirements, architectural elements, and behavioral descriptions, as well as artifact dependencies, failure conditions, and failure rates. This enables the engineer to access data required for a particular safety assessment directly by importing the data representation of the supplier's subrepository or, alternatively, by using the initial manufacturer's specification. Consequently, the repository supports the development life cycle by guiding the safety assessment process and by simultaneously ensuring that relevant design information is provided by the suppliers at each design interaction.

This approach guarantees that each supplier receives all data required for subsystem development and assessment tasks. The iterative re-integration of lower level design and analysis information into the repository is provided by contracts that have been defined according to the dependencies of design artifacts [Engel et al. 2008]. We realize a seamless and structured safety assessment by systematically executing the safety analyses suggested by ARP-4754.

13.3.3 Logical Viewpoint in the Avionics Domain

The purpose of the logical viewpoint is to show the logical architecture of the system under development (SUD). The logical architecture describes the logical components, their subcomponents, and their connecting interfaces.

Activities in the logical viewpoint

Avionics products typically have a very long life time of 30 years and more, therefore it is very important to define architectures that support

the implementation of new functions and the incorporation of new technologies in order to overcome obsolescence problems. The logical architecture provides the right level of abstraction for this purpose. The logical architecture is the first step towards a technical solution without constraining a concrete realization of system elements too early. The avionics group specified the following activities to describe the logical architecture of the system element of this viewpoint:

- Derive logical components from the functions of the functional architecture by grouping coherent functions and allocating these groups to logical components. This results in the logical architecture with its structure.
- Define the states and modes for logical components.
- Functions are linked to requirements and therefore the logical elements are linked to the requirements as well.
- Define the logical interfaces that connect the logical components. The interfaces between function groups are the candidates of the interfaces between the logical components.

Case study: Logical viewpoint

Fig. 13-11 shows the logical architecture of the situation awareness system as part of the logical viewpoint that has been developed from the functional architecture. Functions and interfaces have been mapped to logical components. The definition of logical components is the first step towards the technical realization. In addition, it is used to perform architectural analysis e.g., an integrated fault tree analysis on the system model. This analysis is described in Section 13.3.4.

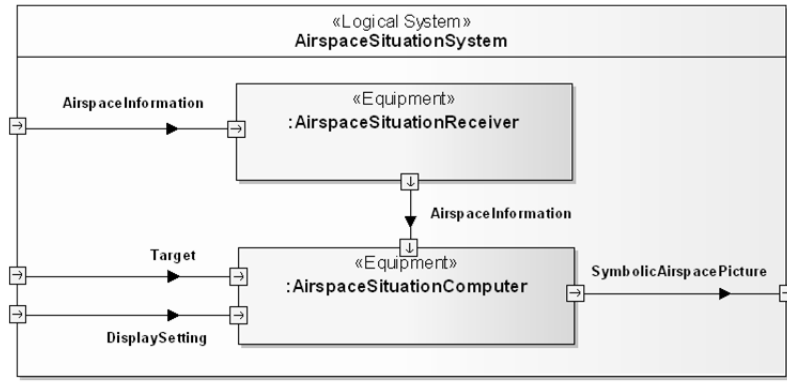


Fig. 13-11 *Logical viewpoint*

13.3.4 Technical Viewpoint in the Avionics Domain

Activities in the technical viewpoint

The purpose of the technical viewpoint is to show the technical architecture of the SUD. The technical architecture describes the technical components and their connecting interfaces with enough detail for their realization. The avionics group specified the following activities to describe the technical architecture of the system element of this viewpoint:

- Define the realization of the logical components and their interfaces.
- Perform trade-off analysis to define the best realization of the technical components described by this viewpoint.
- Logical components are linked to requirements and therefore the technical components are linked to the requirements as well.
- Perform a fault tree analysis (FTA) and failure mode and effect analysis (FMEA) to verify that safety objectives have been met by the technical architecture.

Case study: Technical viewpoint

Fig. 13-12 shows how the technical components are linked to the logical components and how the logical components are linked to functions.

Fig. 13-13 shows the technical architecture of the technical component (equipment) *Airspace Situation Computer* (ASC) as part of the technical viewpoint that has been developed from the logical architecture after the trade-off analysis. Logical components and interfaces have been mapped to technical components. The definition of technical components is the next step towards the technical realization.

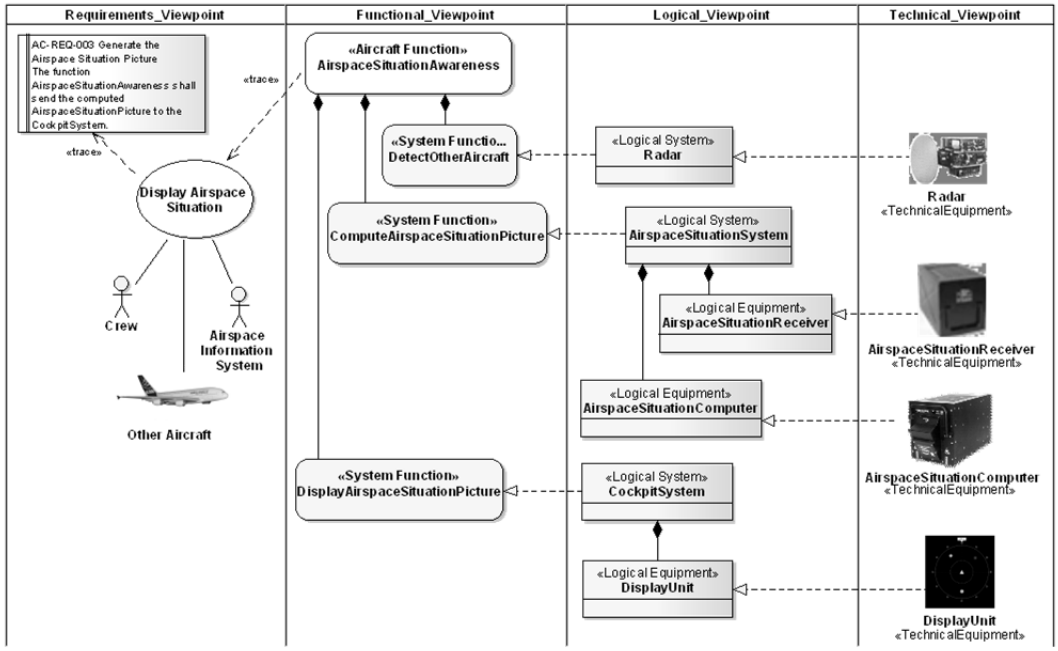


Fig. 13-12 *Transition from the logical to the technical viewpoint*

The technical components are modeled as parts and their interfaces as flow ports.

Case study: Safety

Safety is of paramount importance

For safety-critical systems in the avionics domain, a safety analysis is required to demonstrate that the specified safety requirements have been met. Today, the design and safety artifacts are produced by separate teams working on different repositories for the design and safety models. This has the significant disadvantage of creating redundant artifacts with all related drawbacks. Considerable effort is needed to keep these artifacts consistent. Therefore, it was one of the goals of the avionics group to provide a solution to the fault tree analysis (FTA) topic. For the FTA modeling, an integrated repository was defined containing the design and fault tree information. An important step towards the solution is the definition of a common language for the system designer and the safety analysts. A SysML (OMG Systems Modeling Language) profile has been defined that extends the standard SysML version 1.2 [OMG 2010] notation and semantic with the necessary language elements to express fault tree information. The common notation and integrated

repository allow engineers of both disciplines to work concurrently on the same system model. Today, the SysML modeling tools do not support the computation of probabilities needed for quantitative FTA. To bridge this gap, the fault tree model can be exported to fault tree analysis tools that compute the probability of the faults. This concludes that systems and safety engineers can work together on an integrated design and safety model that removes the disadvantages described above.

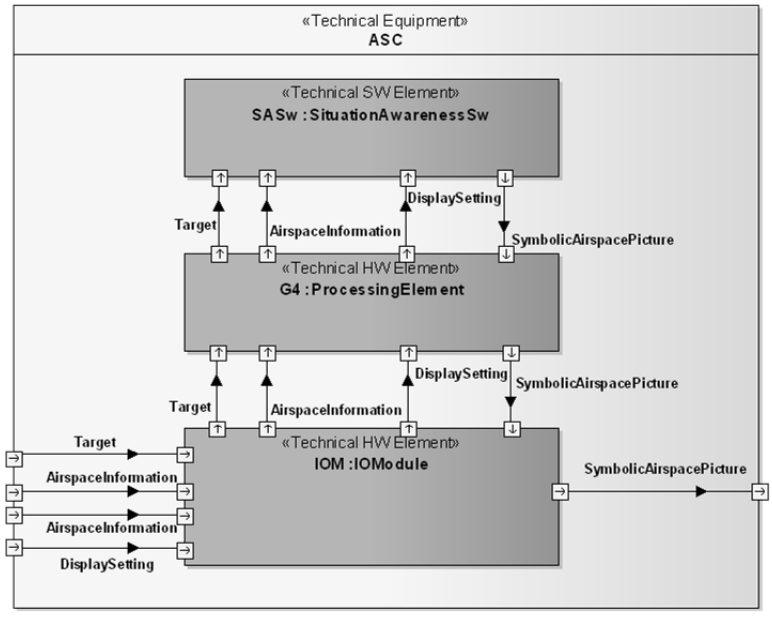


Fig. 13-13 *Technical viewpoint*

Extending SysML to express fault trees

Fig. 13-14 shows the fault tree information of the system element *Airspace Situation Computer* (ASC). It shows the fault tree logic that defines how the faults are propagated from the input ports through the system element ASC to its output ports. The syntax and semantics of the SysML profile shown in Fig. 13-14 are described in Section 8.1. A thorough evaluation study has been conducted to analyze the improvements gained using component fault trees (CFTs) over fault trees. The results show that CFTs have a significant advantage over fault trees, especially for engineers who are not familiar with fault trees.

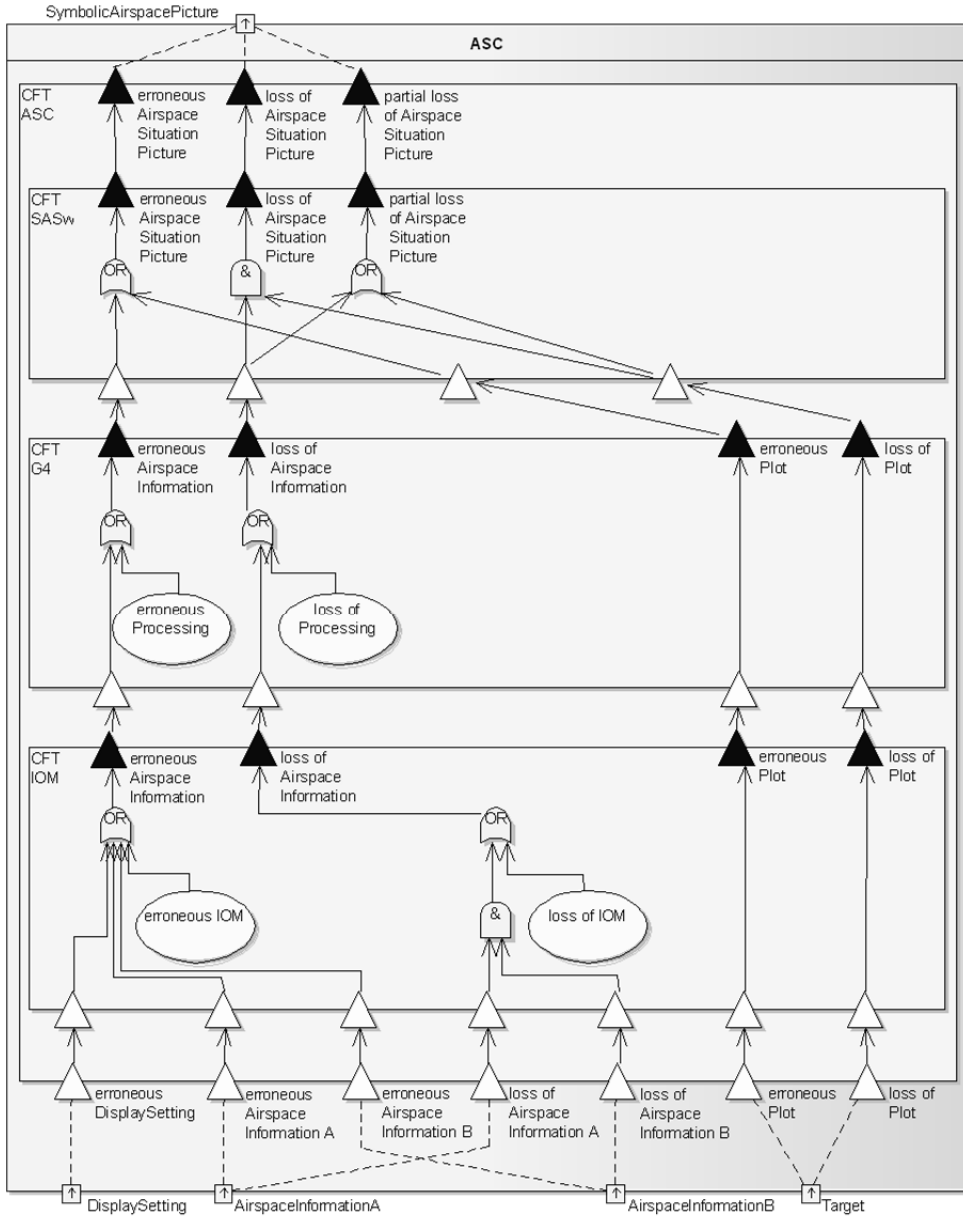


Fig. 13-14 *Component fault tree*

Case study: Verification

The verification activities are among the most important in the avionics industry and create the highest workload in avionics software projects. In other domains, different methods have been developed to reduce the cost of the verification activities, e.g., reuse of the design models to generate test cases and procedures automatically. Motivated by the need to avoid the risk of failures and specified in the strict development process in particular concerning the independence principle required by the avionics standards such as RTCA DO-178B, such widely used approaches are not allowed in the avionics domain.

Reducing verification costs: automation by considering avionics standards

The objective of the investigations made regarding this topic is to determine which activities for the different tasks of the verification process could be automated in the light of the avionics standards and processes. The verification process does more than just execute tests. Other tasks are the development of test cases and procedures, the execution of reviews (for design, source code, test cases, and procedures), or a code analysis. Criteria have been developed for selecting appropriate methods, such as a standardized requirements interface or a dedicated test model, based on their suitability for an automated verification process.

Proceedings for automation in the verification process activities

Methods have been defined in order to achieve automation of the verification process activities according to the avionics standards, such as for test case selection for high-level and low-level requirements, test procedure development, test management, test execution via an automatic test sequencer, generation of test reports, etc. with careful attention given to the concern of tool qualifications. In SPES 2020, the work focused mainly on:

- Automation of the test procedure activities, including generation and verification, are responsible for the highest workload within the avionics verification process.
- Test management automation, which has to facilitate an efficient supervision of the whole verification process and therefore enable the identification and the improvement of inefficient activities.

Test procedure generation

Software tools for test generation

Due to the required independence, a dedicated model must be used as basis for the test procedure generation. To use the existing test cases for

this purpose, they have to be enhanced with keywords that specify signal properties. A software tool has been created to generate test procedures using the test cases description files. It is built upon a dictionary of available signals and their attributes (e.g., unit of measurement, signal direction, range). While parsing each enhanced test case during the test procedure generation, the software tool performs an early and implicit validation of the test case (including checks such as signal existence in the dictionary, specified range, adequate direction, etc.). Additionally, the software tool generates the test precondition and postcondition commands to configure the test environment. With the usage of the software tool, it is possible to generate complete or, for complex test cases, almost complete test procedures. The software architecture of the test procedure generation tool is based on the principle of loose coupling, transforming the test model into an intermediate language before generating a test procedure in the target test sequencer syntax out of the intermediate language. Therefore, it is possible to support additional test models/sequencers with significantly less effort than in conventional procedures.

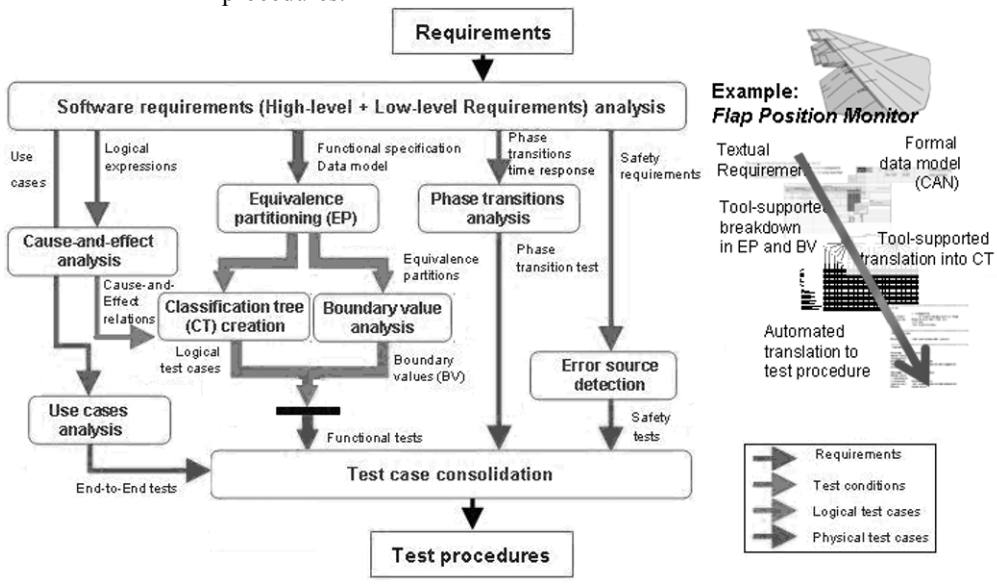


Fig. 13-15 *Test procedure generation: methods and proceedings analyzed*

Test management

One essential precondition for every successful method usage in the verification process is the use of suitable mechanisms to manage all required activities as specified in the applicable standards. Thus, the

determination of mechanisms to set the dependency and sequence of tests, as well as the possibility to automate parts of the management of the verification process, such as the generation of metrics to determine the test fulfillment, were both important parts of the evaluation.

Conclusion and outlook

The avionics standards impose several restrictions regarding the automation of verification activities. The methods investigated and automation achieved take into account all tasks of the verification process and the supporting activities. The activities examined cover the avionics software verification process including test management, preparation/execution of dynamic (e.g., cause and effect) and static (e.g., review) tests, and possibilities for generating test reports automatically. The analysis also addresses the need to perform RTCA DO-178B/EUROCAE ED-12B-compliant qualification of tools that have the potential to reduce the effort needed for the activities or processes required by this standard. The automatic generation of test cases with the current poor standardization of requirements description used in the avionics domain appears to be not feasible at present. It is almost impossible to extract reliable information from natural or weakly formalized languages. For this point, the standardized modeling languages popular in other industry domains, such as SysML, would be useful, in particular when developing an extended syntax including all the necessary attributes in the requirements.

13.4 Summary

Closing the gap between systems and software modeling

The results of the avionics group regarding the research activities in the SPES 2020 program can be summarized as follows.

The methods developed using abstraction layers and viewpoints helped to close existing gaps in the systems and software modeling practice in the avionics domain, e.g., a clear understanding of how to refine and to decompose complex systems and describe the relationship between the design artifacts created. The avionics case study was used to evaluate this concept. Results of this evaluation have been used to refine the model-based systems and software engineering guidelines in the avionics domain. The knowledge gained about the SPES 2020 modeling method has been used by the avionics group to refine its in-house systems and software modeling guidelines.

The SPES 2020 requirements modeling technique has been used for a much better understanding of the system context and the goals of the

requirements defined in the case study. The modeling of the context and the goals provide a good basis for structuring the requirements. It was also recognized that the modeling tools available do not yet fully support the applied method.

The integrated design and safety modeling showed that systems and software engineers can work seamlessly on the same model. The empiric evaluation performed showed that engineers not trained in fault tree analysis have less trouble understanding the CFT notation than the fault tree notation used in separated safety models. Thus, the CFT and the integrated model allow a better cooperation between design and safety teams.

Furthermore, the concept for the integration of safety cases as argumentation support for the certification authority showed a possible future workflow towards automated certification of safety-critical systems in the aviation industry. The industrialization of this novel approach will be investigated in future research projects.

The results achieved for the verification part of the avionics case study showed that parts of the verification cases can be generated out of formalized requirements.

It has been proven that the definition of the avionics case study was essential for the success of the results achieved. It provided a clear basis for the communication of the problems to be solved in the avionics domain. Therefore, developing such a case study is recommended as a best practice for future research programs. However, it shall be noted that the effort to create such a case study should not be underestimated.

Important steps towards the solution of the problems expressed

In conclusion, we can say that the cooperation of the academics and avionics group in the SPES 2020 program was very beneficial to both sides. It enabled both parties to make important steps towards the solution to the problems expressed in the avionics case study.

13.5 References

[Domis and Trapp 2009] D. Domis, M. Trapp: Component-based abstraction in fault tree analysis. In: Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP 2009). DOI: 10.1007/978-3-642-04468-7_24.

[Engel et al. 2008] A. Engel, M. Winokur, G. Döhmen, M. Einzmann: Assumptions / promises - Shifting the paradigm in systems-engineering. In: Proceedings of INCOSE 2008, 2008.

[OMG 2010] Object Management Group: OMG Systems Modeling Language™ (OMG SysML) Language Specification v1.2. OMG Document Number: formal/2010-06-02.

[Waßmuth and Stilkerich 2011] Waßmuth, M., Stilkerich, S. C., Lübbers, E.: Distributed safety assessment for airborne systems. In: International Workshop on Security and Dependability for Resource Constrained Embedded Systems, Italy, 2011.

Dr. Friedrich-W. Fasse
Christian Glomb
Johannes Grünbauer
André Heuer
Martin Klaus
Dr. Richard Kuntschke
Prof. Dr. Michael Laskowski
Dr. Thorsten Weyer

14

Application and Evaluation in the Energy Domain

The activities in the energy domain focus on the model-based development of embedded systems for smart grids. In this context, the domain investigates typical use cases, analyzes the requirements placed on embedded systems within the energy domain, identifies and evaluates possible modeling approaches, extends and evaluates the SPES requirements viewpoint within the energy domain, and actively applies and evaluates the SPES modeling framework to model, develop, implement, and evaluate a smart grid simulator.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_14,

14.1 Overview: Application Domain Energy

Importance of the energy domain

Energy providers in Germany, including grid operators, play an essential role in securing the value chain of almost all business sectors as well as supplying private households with energy. Electricity utility companies, for example, represent a significant factor in the German economy with their 132,000 employees, generating a capacity of approximately 561 billion kWh, and their overall investment of more than €10 billion. Given the fact that 68% of the electricity generated is used for production processes in industry and by the commercial, trade, and services sector, the significance of electricity utilities for the German economy is abundantly clear. The primary importance of the energy domain is further underlined by the fact that German electricity grid operators are responsible for a transmission network that is 1.8 million kilometers long, yet has an average annual outage period of only 18 minutes — making it the most reliable electricity grid throughout Europe (99.9965% reliability [BDEW 2010]).

Importance of embedded systems in the energy domain

Embedded systems already represent an integral part of the technical systems used in the energy market and perform clearly defined roles such as managing and controlling power plants. However, current trends in the energy sector indicate that the demands placed on embedded systems in terms of their performance capability within the energy domain will significantly increase in the future. Millions of embedded systems are expected to be deployed, e.g., in ICT gateways (one per household) at reasonable costs (comparable to that of a DSL router) with an expected lifetime of up to 25 years.

Smart grids

For a number of years now, there has been a massive trend towards smart grids. The term smart grid has arisen from the modernization of existing grids to enable them to cope with rising demands for a secure and efficient electricity supply both now and in the future. This trend is driven by a range of political, economic, and environmental goals that will also have a direct impact on the information and communication technology used to manage the future electricity grid. For instance, the expansion of renewable energies such as wind power, photovoltaic, and biogas has been increasingly pursued over the past few years to meet climate change goals and to conserve resources while providing a sustainable energy supply. This has led to an increased number of smaller, decentralized generation units in addition to Germany's existing large-scale power stations. Moreover, the need to use energy more

efficiently has led to increased transparency around energy consumption and to a smarter use of electrical devices.

Embedded systems in smart grids

These trends have given rise to new demands concerning the functionality and use of embedded systems that, in the smart grids of the future, will be used not only for control but also for information and communication purposes. For instance, embedded systems will soon have to be capable of offsetting the increase in dynamic interrelations caused by increasing volumes of power being fed onto the grid from renewable energy sources, automatically switching residential electrical consumers on and off grid, communicating a host of different meter readings, and supporting new business models through intelligent networking of smart meters and smart home technologies.

Only the introduction of more advanced embedded systems will meet the technical demands for ever-more sophisticated control, information, and communication tasks within the smart grid, as well as the ability to generate appropriate added value in the form of new business models. Embedded systems will significantly contribute to the task of reaching those ambitious political, economic, and environmental goals.

Challenges within the energy domain

Future smart grids will have a degree of complexity that bears little resemblance to any other known system. As a result, the development of embedded systems for smart grids in the energy domain poses specific challenges that cannot be found in other domains.

The smart grid stands out for the extreme complexity of its structure and interconnectivity, combined with the highly dynamic structure and behavior of its systems. Furthermore, an extremely large number of embedded systems with massively distributed components will have to be used within the smart grid. In addition, embedded systems within the smart grid will not only have to be integrated in the technical environment but also in the business processes of the energy companies. Eventually, due to the sheer size and complexity of smart grids, a host of technical decisions and solutions devised during the development process will be virtually irreversible after implementation (“one-shot scenarios”).

Currently, little experience is available in developing systems comparable with the complexity of such future smart grids. As a consequence, the energy domain focuses on devising efficient development methods for massively distributed systems with embedded components in smart grids. In particular, this comprises gaining an understanding of the specific requirements of the energy domain, grasping the extent of the inherent problems, and then designing, adapting, and fine-tuning some targeted methods for developing and mastering embedded systems in the context of smart grids.

14.2 Evaluation Strategy in the Energy Domain

14.2.1 Modeling Languages, Theories, and Tools

Modeling concepts: applicability and benefit

Early tasks in the SPES project in the energy domain comprised the evaluation of several modeling languages, theories, and tools with respect to their applicability within the energy domain. The evaluation focused primarily on the extent to which these modeling concepts help to solve the challenges mentioned in Section 14.1. The evaluation comprised theoretical analyses as well as some limited practical applications of modeling concepts to the use cases considered within the energy domain. The modeling concepts evaluated include the different viewpoints of the SPES modeling framework, especially the requirements viewpoint (see Chapters 4 through 7).

14.2.2 Requirements Engineering

Requirements process model :applicability and benefit

The objective of the requirements engineering evaluation activities was the practical application of the requirements viewpoint and the corresponding requirements process model (see Section 4.4). The aim was to obtain empirical evidence of the benefit of the requirements engineering approach in the energy domain, in particular concerning issues such as the reduction of complexity and the integration of a business process view.

The method was used in four workshops with experts from the energy domain. The evaluation was carried out in close cooperation with academic partners and developers of the SPES requirements viewpoint (see Chapter 4) by means of ex post facto design questionnaires.

14.2.3 Integrated Smart Grid Development

AutoFOCUS 3: capability and limitations

In close cooperation with academic partners, experts from the energy domain assessed the practicability of the integrated development of embedded systems in smart grids by implementing a demonstrator. The focus was mainly on the feasibility of modeling domain-specific aspects, including dynamic system structure and behavior, a large number of components, as well as code generation and subsequent deployment to embedded hardware components using the AutoFOCUS 3 tool. Furthermore, the viewpoints of the SPES methodology (see Chapters 4 through 7) were taken into account. Based on the results, the energy domain suggested improvements to the tool as well as to the underlying

methodology. As described in Section 14.2.3, a smart grid simulator was also developed and evaluated.

14.3 Evaluation Activities and Results

This section illustrates how the SPES modeling framework has been tailored for the specific needs of the energy domain and how the SPES modeling framework has been evaluated.

14.3.1 Incorporating Activities within the SPES Modeling Framework

The SPES metamodel in the energy domain

The work of the energy domain was based on the SPES modeling framework (see Chapter 3). Concepts defined within the SPES modeling framework for addressing the degree of complexity were taken into account in the sense that they were partly adopted or suitably adapted to the special requirements of the energy domain. This includes the use of generalized artifacts, the definition of logical (abstraction) levels, and the component-based design of embedded systems.

In the *Energiemeister* case study, for example, analysis was conducted (from an operational point of view) of how components within the smart grid can be integrated into the system landscape of an energy utility. In this case, the requirements engineering activity for the system under development was jointly carried out through to the semiformal model stage (use cases, tables). Furthermore, the logical viewpoint (see Chapter 6) described in the SPES modeling framework was taken into account during modeling using the AutoFOCUS 3 tool (structural diagrams of the system). In addition, AutoFOCUS 3 was used to compile a model-based description of the technical architecture.

The requirements process model evaluated by the energy domain is also closely related to the SPES modeling framework. This approach defines a process whereby requirements engineering artifacts and the architecture of a system can be co-designed. The requirements process model uses artifacts from the requirements viewpoint (goals, scenarios, and solution-oriented requirements) and various abstraction layers derived from the SPES modeling framework. Furthermore, the framework specifies a structured way of incorporating the abstraction layers into the requirements engineering activity.

14.3.2 Joint Partner Activities

Overview of project activities

In the first half of the project, work within the energy domain established the basic foundations by identifying specific case studies from the energy domain, examining requirements, and evaluating modeling theories in preparation for developing a suitable model.

In the second half of the project, work focused primarily on putting the fundamentals established in the first half of the project into practice for evaluation purposes.

For instance, some of the work in the second half of the project involved empirical testing of the approaches conducted in SPES 2020 for developing and mastering embedded systems. Other work involved creating a simulated environment for the purpose of analyzing the features and behavior of embedded systems within a smart grid. The main focus of this work was on planning, developing, implementing, and evaluating a smart grid simulator using the SPES modeling framework as described in Part II.

Case studies

Case studies

In SPES 2020, a case study named *Smart Grid* was developed. The goal of this study was the aggregation and integration of case studies developed by the individual partners within SPES. The individual case studies take views on the subject *Smart Grid* from different perspectives (prosumer/ICT gateways, virtual power plant, *Energiemeister*) that are closely tied together. Thus, the case studies focused on a scenario that combines all case studies and mainly considers the activities of monitoring and controlling distributed energy generators and energy consumers within a smart grid.

Requirements for system modeling

Requirements analysis

As discussed in Section 14.1, challenges within the energy domain comprise, amongst other things, massively distributed components, the dynamic integration of new components into the smart grid preserving stability, and the reliability of energy supplies. This raises characteristic requirements for system modeling such as support of dynamic system structures with a huge number of components, support of stochastic processes (behavior of components and of communication systems), scalability of modeling, consideration of a business model, support of an event concept, and modeling of components including component properties and relationships.

Identification and evaluation of modeling languages, theories, and tools concerning the SPES modeling framework

Evaluation of modeling concepts

During modeling of the use cases in the energy domain, various modeling languages, theories, and tools were used. These include the Unified Modeling Language (UML) [OMG 2010] and the UML modeling tool Enterprise Architect, the functional and logical viewpoints (see Chapters 5 and 6) together with the AutoFOCUS 3 tool, and the requirements viewpoint together with the corresponding requirements process model (see Section 4.4). The latter two are part of the SPES methodology as introduced in Part II.

UML and Enterprise Architect

As far as UML is concerned, its generic approach provides for broad applicability. At the same time, however, specific needs of specialized application domains are neglected. In the energy domain, the missing concepts for modeling requirements and dynamics in system structure and system behavior proved problematic. As far as Enterprise Architect is concerned, it proved perfectly usable within the boundaries of UML and even offers an approach for adding requirements in the models.

Functional/logical viewpoints and AutoFOCUS 3

The functional and logical viewpoints of the SPES methodology proved usable in the energy domain. Similar to UML, however, the approach lacks possibilities for modeling dynamics in system structure and system behavior. Further, it assumes a global clock, which is an unrealistic assumption in a system with massively distributed components such as a smart grid. There are, however, possibilities to extend the SPES methodology with support for the aspects mentioned, and the energy domain has thus drawn attention to these issues within the SPES project. AutoFOCUS 3 was used for modeling and automatic code generation in the energy domain during smart grid simulator development. Here, the issue of state proliferation when using complex models was observed.

Requirements process model

The SPES requirements viewpoint and the corresponding requirements process model have also been used actively in the energy domain and have proven useful. Also, the methodology has been extended by integrating an additional business process layer as suggested by the energy domain.

Practical evaluation of the SPES methodology

Evaluation of the requirements process model

To evaluate the benefit of the requirements viewpoint in the energy domain, the approach was applied practically to the initial requirements analysis of two specific development projects in the domain. The positive results indicate a great benefit of the corresponding requirements engineering methodology during the initial requirements analysis for the

development of embedded systems in the energy domain. For a detailed description of the evaluation and its results, please refer to Section 14.4.1.

Development and evaluation of a smart grid simulator

Further work within the energy domain investigated the integrated development process and subsequent evaluation of a smart grid simulator using the SPES modeling framework. In addition, the evaluation focused on the applicability of the resulting simulator as a design tool within the smart grid development process. Section 14.4.2 describes the details of smart grid system modeling and development in practice.

14.4 Exemplary Evaluation Activities in Detail

In-depth examples of evaluation activities in the energy domain

As an in-depth example of evaluation within the energy domain, the following sections present the activities concerning requirements engineering and smart grid simulator development. These activities cover the entire smart grid development process. To enable the holistic evaluation of the entire development process within the limited time frame of the SPES 2020 project, the requirements engineering activities and the smart grid simulator activities were split and elaborated on in parallel during the project.

14.4.1 Requirements Engineering in the Energy Domain

Evaluation of the requirements process model

Part of the requirements engineering work in the energy domain involved empirical evaluation of the requirements viewpoint developed in the SPES 2020 project for the purpose of developing and engineering embedded systems.

Subject and objective of the evaluation

Evaluation subject

The subject of the evaluation was the practical application of the requirements engineering methodology to the initial requirements analysis of two specific development projects in the energy domain:

- Wind heating — negative operating reserve
- Data management in smart grids

The method was used in four workshops with a total of 18 experts (e.g., chemists, electrical and mechanical engineers, economists) from various divisions (e.g., sales, grids portfolio management, product management).

It is worth mentioning that the requirements viewpoint, among other things, also incorporates a business process layer (see Fig. 14-1). In this way, the requirements engineering methodology meets one key

requirement of the SPES 2020 project — to explicitly take into account defined/existing business processes when developing any embedded systems.

Evaluation objective

The aim of evaluating the requirements viewpoint within the energy domain was to transfer empirical evidence about the benefit of the requirements engineering methodology to development projects of embedded systems in the energy domain. This evaluation goal was measured, or rather assessed, in more detail against the following factors:

1. Additional insights gained from users of the methodology
2. High-value findings for future work
3. Positive assessment of the requirements engineering methodology
4. Improved self-assessment with respect to the development project

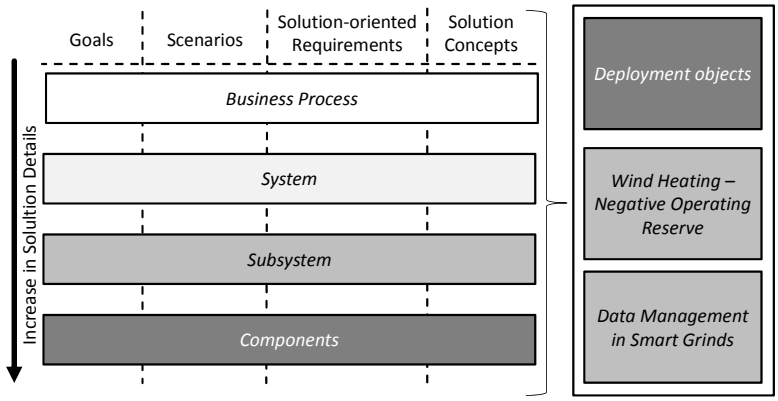


Fig. 14-1 *Application of the requirements viewpoint for development projects within the energy domain*

An ex post facto design with pre- and post-measurement via questionnaires ❶-❷ plus additional deferred post-measurement via questionnaire ❸ two weeks after the workshops was used to measure the factors 1, 3, and 4 above (cf. Fig. 14-2). The design was developed in collaboration with experts from the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern. Factor 2 was evaluated by an objective expert who did not take part in the above-mentioned workshops.

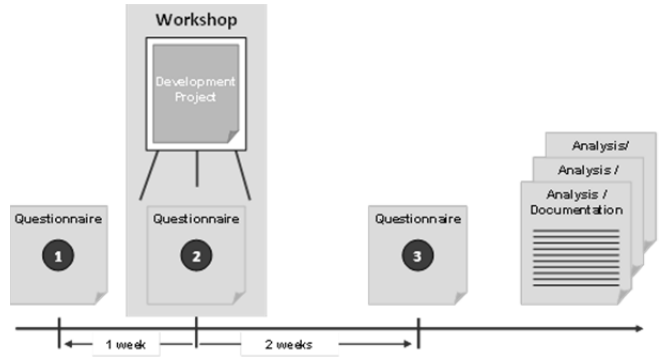


Fig. 14-2 *Ex post facto design to evaluate the use of the requirements viewpoint within the energy domain*

Evaluation procedure

Evaluation procedure

Evaluation of the requirements engineering methodology was undertaken by asking specific questions related to embedded systems about the above-mentioned energy domain development projects. The questions were addressed during workshops with experts from the energy domain, where the requirements viewpoint was applied and its impact was assessed. The evaluation procedure consisted of the following seven steps:

Evaluation steps

- Selection of the evaluation project and participants
- Initial evaluation prior to the workshop for the purpose of self-assessment by the participants in relation to the development projects
- Use of the requirements viewpoint in the workshop
- Second evaluation immediately after the workshop (insights gained, assessment of the method)
- Scrutiny of the results by an objective expert (person with the expertise of a specialist who did not take part in the workshop)
- Third evaluation after the workshop
- Documentation of the findings

Evaluation results

Benefit of the requirements viewpoint in the energy domain

The evaluation of the requirements viewpoint in the context of the above-mentioned development projects delivered some initial findings about the applicability and benefit of the requirements viewpoint for developing embedded systems in the energy domain. The following findings represent a summary of the evaluation completed as a result of the workshops. Since one development project is still highly confidential, the findings are depicted in abstract terms only.

Requirements process model contributions

With regard to the substantive aspects, the requirements viewpoint has made a significant contribution towards structuring the problems and reducing complexity:

- Abstraction stages/levels are evident (business process, system, subsystem, and components)
- Goals at various abstraction stages/levels are defined
- Scenarios are modeled
- Solution-oriented requirements are identified
- Initial approaches for a problem-solving concept are developed

Relationship to the SPES modeling framework

The close relationship of the applied methodology to the SPES modeling framework is therefore clear: the specific requirements viewpoint that was used within the two development projects defines an appropriate adaption of the SPES modeling framework to the energy domain, for example, by using specific abstraction layers and solution-oriented requirements artifacts of the requirements viewpoint.

Requirements viewpoint benefits

By abstracting and/or refining the points of view and describing elements in varying degrees of detail, a successful focus on key aspects was achieved with a view to possible architecture models. Consequently, we can confirm that there is a great benefit to be gained, in terms of reducing complexity, from applying the requirements viewpoint to designing embedded systems for the energy domain. This is also confirmed by the evaluation of the requirements methodology, which points to an overwhelmingly positive outcome.

Assessment of workshop participants

The participants showed a high degree of consensus on the insights gained, the benefit of the method, and the relevance of the subject matter. Those involved in the workshops gained new knowledge by applying the method, and also evaluated the requirements engineering methodology as positive. Above all, the participants viewed the structured process used to analyze problems as positive. They also found the use of goals, scenarios, and abstraction layers to be helpful. Furthermore, they commented that their own grasp of the subject matter improved as a result. In particular, the requirements viewpoint and the corresponding methodology helped participants to gain a better in-depth understanding of the subject matter.

Assessment of the objective expert

These statements are supported by the assessment of the objective expert, who perceived the application of the requirements process model positively.

Potential for improvement

The participants involved identified a potential for improving the requirements viewpoint with respect to autonomous application of the corresponding methodology. According to the participants, the use of a trained moderator is essential. Furthermore, the participants considered it

difficult to transfer the (independently) applied method to other practical examples. Also, it was noted that the requirements engineering methodology supports a deeper analysis of problems but not an analysis "on the fly."

Conclusion

Overall, the requirements viewpoint and the corresponding methodology prove to be a beneficial approach for developing embedded systems in the energy domain. With regard to future applications of the approach, the identified potential for improvement should be taken into account.

14.4.2 Smart Grid System Modeling and Development

Development of an integrated smart grid simulator

Smart grid simulator development: motivation and challenges

Smart grids are high-grade dynamic systems with a large number of widely distributed components. Due to their size and complexity, a formal verification of system properties is not feasible in practice. Thus, system simulation has gained importance as a major part of the integrated model-based development process. In the energy domain, a smart grid simulator addresses the following challenges that can also be found in comparable systems: (a) uneven balance of energy generation and consumption, (b) lack of stability of the overall system, (c) vulnerability of the system due to dynamic changes of the behavior of consumers and producers, (d) vulnerability of the system due to faults or failures of communication and/or components, (e) system stability with a huge number of components, and (f) limitation of system optimization resources.

To keep pace with these challenges and to avoid problems while developing the overall system, the proof of correctness of the concept must happen in an early stage of the development process.

Actuating variables

The behavior of the real system can be deduced based on the actual values for the actuating variables of the simulation and the corresponding simulation results. The relevant actuating variables reflect the properties of the main constituents of a smart grid: the electric supply network, the communication network, and such (local and decentralized) smart grid components that are connected via the networks (see Fig. 14-3).

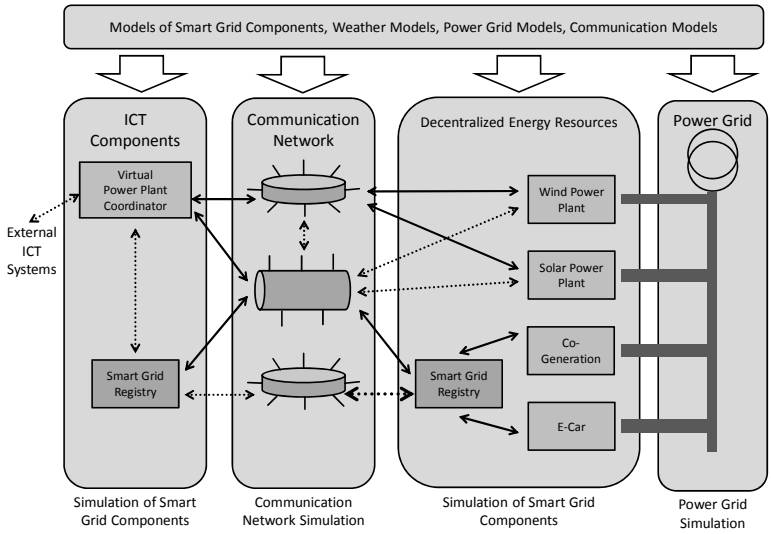


Fig. 14-3 *Overview of simulation components*

The simulation environment for investigating the behavior and interaction of smart grid components consists of two layers: on the *simulator layer* the VPP topology, i.e., the number of components and the system structure can be configured. The second layer contains the *VPP coordinator* and the decentralized energy resources attached. These components can be parameterized using, e.g., consumption and weather profiles. Components can be added and removed dynamically. Also, communication failures can be simulated. The following summarizes the dependencies between the simulations and the simulated components:

- *VPP coordinator.* This is the centralized controlling, monitoring, and optimization system for the decentralized plants forming a virtual power plant. Further components offer the registration and administration of decentralized plants.
- *Simulation of the communication network.* The communication network has to be simulated with regards to (a) bandwidth, (b) varying number of participants, (c) latency, and (d) reliability.
- *Decentralized energy resources* are, depending on the configuration, (a) producers, (b) consumers, and (c) prosumers that entail both characteristics (a) and (b). Entities that reflect those characteristics, such as wind engines, solar modules, biogas plants, co-generators, or eCars, can be integrated.
- *Simulation of the electric supply network.* The actuating variables of the electric supply network are (a) cable cross-section, (b) voltage

level, (c) concurrency factor, (d) safety concept, and (e) concepts for redundancy (alternative cableways).

- *Simulation of smart grid components.* The actuating variables of the various smart grid components comprise, depending on the nature of the component, (a) the number of parallel communication connections of a single component, (b) the size of the internal data storage, (c) the physical component model, and (d) the predictability and fluctuations of generated or consumed power levels.

Power grid simulation

PSS™ NETOMAC power system simulator

The Siemens Power System Simulation (PSS™) software suite was chosen for the integrated smart grid simulator. PSS™ is a set of software tools used by many European power companies, as well as universities, for the design and simulation of electric power systems. NETOMAC is the mathematical solver within PSS™.

NETOMAC sets up the system admittance matrix $\underline{Y}$ describing the elements of the modeled power grid with a set of differential equations. The differential equation system is solved for each calculation time step of a configurable duration using the difference conductance approach [Kulicke 1981]. Thereby, a continuous numerical integration is performed using the trapeze method.

Simulation time can be synchronized to a real-time clock, e.g., the PC clock, allowing hardware-in-the-loop tests as well as connections to control networks or components, e.g., ICT gateways, and to real clients such as a virtual power plant control center used to control components of the simulated power network as in our scenario.

Communication network simulation

NS-3 network simulator

The communication network simulator NS-3 [NSF 2011] is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use.

NS-3 offers the possibility to run in emulation mode, i.e., to simulate a communication network in real-time, capturing and outputting live IP traffic from and to the interface cards and conveying it through the network modeled. In real-time mode, NS-3 can be used for hardware-in-the-loop tests along with NETOMAC.

Co-simulation

Combined communication network and power grid simulation

Co-simulation aims at studying the interdependencies of smart grid components. Thus, an environment for combined or co-simulation comprising the components as described above was set up.

Pure power grid simulators are mainly built to analyze the electrical characteristics and behavior of a power grid. They are restricted with respect to capabilities for adding controllers to the power network, e.g., to model generators or protection devices. Also, communication between distributed controllers is not in the focus of such tools. However, in most cases, the user may add external control logic that can access power grid parameters (voltages, currents, etc.) during simulation to the power grid simulator. In the same way, model parameters such as machine parameters or breaker settings can be influenced by external logic. Instead of arbitrarily embedding control logic into the power grid model, our approach identifies a defined set of standard interaction points to get measurement values or to set machine parameters.

To set up a co-simulation, monitoring and control applications can interact with the power network using a Java interface. The communication between these applications and the energy network simulator corresponds basically to real power system communication using [IEC 61850] compliant devices.

The properties of the communication network used to carry the commands and measurements gain importance as the number of communicating devices increases. When a shared infrastructure such as the Internet or power line communication is used, it is especially important to understand the influence of communication network properties on the behavior of the power grid.

Simulation & synchronization

Besides emulation, non-real-time operation, i.e., simulation is also possible. Synchronization in simulation mode is based on the idea of letting simulators run independently for a defined simulation step, which may last for a longer or shorter time than the real time, and re-synchronize them after the simulation step has been performed.

Smart grid simulator evaluation

Evaluation goal

The primary evaluation goal concerning the smart grid simulator consists of proving its applicability as a smart grid development and design tool fitting well into the smart grid development process. To prove applicability, the evaluation investigates whether and to what extent the simulator fulfills the requirements listed below.

Simulator requirements

The smart grid simulator shall allow for examining system and grid stability. Example scenarios shall be investigated for the possibility of a

stable continuous operation. Relevant aspects include dimensioning of smart grids, analysis of the behavior of systems with a large number of components (more than 10,000), and optimized power allocation. The simulator shall be able to simulate dynamic processes in system behavior. Dynamic processes comprise faults such as failures of components or communication lines, as well as controlled processes such as fluctuations in power generation and attaching or detaching producers or consumers.

Reference scenario

The evaluation employs a reference scenario modeling a low voltage grid area based on real-world power grid data provided by RWE and assumes a 100% PV penetration. The corresponding communication network is assumed to have the same structure as the power grid and the properties of either a DSL access network (ideal case) or a power line communication (worst case).

Simulation results

Simulation results indicate that production and generation schedules computed by the VPP coordinator based exclusively on economic constraints lead to a temporary overload in the power grid resulting in voltage boundary violations.

Thus, either a reconfiguration of the grid area is necessary, i.e., an installation of additional or thicker cables, or an appropriate control of both producers and generators has to be introduced. In the latter case, the simulator helps in selecting and tuning possible countermeasures. These include: injection of reactive power from the PV units, switching on additional effective loads, and changing the tap at the local transformer station.

The introduction of control mechanisms based on measurement values from client-side meters requires an available communication network. Simulation results show that control has to deal with incomplete information in the worst case due to packet loss and a delay in the range of minutes, whereas in the ideal case, prioritization of control traffic over user-initiated traffic such as web-browsing offers the possibility of fast control since all measurement values arrive within a few milliseconds.

In summary, the smart grid simulator proved to be a valuable dimensioning and early evaluation tool for smart grid development.

14.5 Summary

Summary

Work in the energy domain within SPES 2020 focused mainly on four aspects as described above: the identification and specification of suitable case studies to analyze the domain and its requirements with respect to the development of embedded systems, the analysis and

evaluation of the SPES modeling framework and corresponding modeling approaches from the perspective of the energy domain, the extension and subsequent evaluation of the requirements viewpoint, and the integrated model-based development of a smart grid simulator.

Case studies

The identification and specification of suitable case studies is especially valuable in the energy domain since the smart grid setting that is in the focus of the domain is very innovative and there is virtually no prior experience with existing systems. Thus, thinking about the possibilities and requirements of embedded software in the context of smart grids provides necessary and valuable insights. The integrated case study concentrates on the issues of monitoring and control in a smart grid. From this perspective, it provides insight into all relevant parts of a smart grid, from virtual power plants to power management to ICT gateways.

The analysis and evaluation of the SPES modeling framework and corresponding modeling approaches, including the modeling of system requirements and concepts, provides an overview of available techniques and tools for software and systems modeling together with an evaluation of their suitability within the energy domain. We can conclude that while most approaches are usable in some way in the energy domain, not every peculiarity of the energy domain is addressed. Hence, further tailoring is necessary. For example, this includes the ability to investigate emergent issues that arise from dynamics in system structure and system behavior. These issues need to be considered in further work within the energy domain.

Requirements process model

Subsequently, the requirements viewpoint has been extended according to the needs and constraints identified in the energy domain by adding a business process layer. Further, the adapted SPES modeling framework has been intensively evaluated in a series of workshops conducted in cooperation with industrial partners. The results of the evaluation show that the SPES modeling framework is well-suited to support the requirements engineering process for the development of embedded systems in the energy domain.

Smart grid simulator

Finally, the complexity of smart grid installations makes the formal analysis of such systems virtually impossible. Therefore, the importance of simulation as a means for designing and evaluating smart grid installations in early stages of the development process increases. In the energy domain, an integrated smart grid simulator has been developed. The SPES modeling framework was successfully applied in this case study and resulted in requirements, functional and logical architecture, and a deployable technical architecture that, in later stages, was successfully used to simulate smart grid installations. Using the SPES

modeling framework, it was possible to develop an integrated simulator that can serve as an additional development tool in the development process of smart grids.

14.6 References

[BDEW 2010] BDEW Bundesverband der Energie- und Wasserwirtschaft e.V. (Hrsg.): Energiemarkt Deutschland. Zahlen und Fakten zur Gas-, Strom- und Fernwärmeversorgung, 2010.

[IEC 61850] IEC International Standard 61850-7-420: Communication Networks and Systems for Power Utility Automation – Part 7-420. International Electrotechnical Commission, 2009.

[Kulicke 1981] B. Kulicke: Simulationsprogramm NETOMAC: Differenzleitverfahren bei kontinuierlichen und diskontinuierlichen Systemen. Siemens Research Reports, Vol. 10 No. 5, Springer, 1981, pp. 299-302.

[NSF 2011] National Science Foundation: Network Simulator NS-3: http://www.nsnam.org/. NSF Planète Group at INRIA, 2011.

[OMG 2010] Object Management Group: OMG Unified Modeling Language™ (OMG UML), Infrastructure v2.3. OMG Document Number: formal/2010-05-03.

14.7 Acknowledgements

The authors wish to thank the following people for their support and their contributions to the work presented in this chapter: Johannes Bergmann and Dr. Jörg Heuer (both of Siemens AG), Dr. Martin Fritzsche and Sascha Schwind (both of TU München), Dieter Heisterberg (RWE Consulting GmbH), Dr. Kim Lauenroth and Dr. Ernst Sikora (both formerly of University of Duisburg-Essen), and Dr. Edmund Simmet and Dr. Alexander Vilbig (both of SWM Services GmbH).

Hendrik Heinze
Dr. Khalid Kallow
Harmut Lackner
Dr. Sadegh Sadeghipour
Prof. Dr. Holger Schlingloff
Salko Tahirbegovic
Dr. Hans-Werner Wiesbrock

15

Application and Evaluation in the Healthcare Domain

This chapter deals with the application of the SPES modeling framework to a medical case study: an extended care system (ECS). This system allows patients with serious heart conditions to stay in their home environment while remaining under constant medical surveillance and assistance. The ECS is typical for future telemedical applications, where body sensors and implanted devices work together with an ambient IT infrastructure to guarantee optimal patient-centered care. Design challenges in this case study are the safety and reliability requirements, interface definitions and architecture of the combined system, as well as meeting regulatory demands for life-supporting subsystems while adding further, not necessarily safety-critical components to the system.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_15,

15.1 Overview: Application Domain Healthcare

In the healthcare domain, software as an integral part of medical equipment and processes is becoming increasingly important. Software-based systems control and influence more and more medical activities, from first aid to rehabilitation. Similar to the other domains in this book, embedded software plays a central role in the innovation and progress of medical technology. Typical devices are large hospital and laboratory apparatuses, surgery appliances for diagnosis and treatment, and patient equipment usually for the mass market. In all these products, more and more of the innovative functions are realized by software.

Current trends in medical systems

The following are some current trends that are determining and are determined by the development of embedded systems in the field (see, e.g., [Glesner et al. 2007] and [Zauner and Schrempf 2009]):

- Interoperability and interconnectedness: As embedded medical systems obtain more and more communication interfaces, new integrated services are becoming possible. Examples are diagnostic implants that transmit vital data directly to the electronic health record, or the integrated operating theater where the surgeon has central control of various devices and displays.
- Telemedicine and telematics: Embedded devices allow not only remote monitoring of the patient's health, but also the provision of real-time interactions between patient and physician. Robot-assisted intervention devices, such as remotely operable catheters, assist in complex, minimally invasive surgery.
- From diagnosis to therapy: Due to increased sensing and processing power, devices that were formerly only able to observe the patient or provide basic services are now capable of accomplishing complex treatment. One example is a pacemaker with sensors monitoring the heart activity that can now help to cure various cardiopathies.
- Ambient assisted living: Systems designed for the support of elderly or handicapped people in their home can reduce healthcare costs and increase the quality of life. Examples range from simple automated pill dispensers to intelligent wheelchairs and intelligent prostheses that restore physical abilities, such as cochlear implants.

Similar to the aerospace domain, most medical devices are safety-critical in the sense that incorrect software could harm the patient. The more critical the application, the more important the role of the embedded control system within the application and the software used to control it.

In contrast to applications in other industrial domains, such as railway or automotive, the full functionality has to be maintained even in the case of a single fault. It is not an option to shut down the device or go into a "fail-safe state." As a consequence, all functions must be implemented redundant and diverse under the condition of real-time parallel signal processing. Redundancy can protect against random (stochastic) faults during the operation, whereas diversity is used to protect against systematic faults in the design. Fault tolerance is often achieved by parallelism in the hardware. As the system must be shown to be resistant even against common mode errors, the redundant parts must be designed to be largely independent.

Due to the increased complexity of the design, conventional development methods are no longer appropriate for producing code that can satisfy these requirements. As shown in Chapter 9 on modeling quality aspects, model-based design can be used to support the development of such highly critical systems. In the medical domain, development tools also have to respect the above-mentioned special requirements regarding functional safety.

Dramatically increased verification and validation efforts

In parallel to the increase in the complexity of systems, the amount of verification and validation in the development has risen dramatically over the last few years. This is mainly due to the following reasons:

- The increase in embedded systems technology allows the measuring, influencing, and controlling of more and more physiological parameters of the human body. Therefore, the systems are used in more and more critical applications. This increase in capabilities leads to an increase in complexity in the validation and certification process.
- The evolution and international harmonization of standards regulates the development processes at a much more finely-grained level than in former times. Therefore, the normative and legal requirements of the development process must be followed and documented to a much higher extent.
- The increased criticality leads to more and more supervision by authorities. In Europe, for example, Class I medical products can be developed and marketed by the manufacturer in their own responsibility, whereas Class III and IV products are completely supervised by an authority. The manufacturer must provide all necessary evidence and show that he conforms to all essential requirements of the respective European directives.
- As a social component, the user expectations of electronic equipment are constantly rising, both with respect to their functionality and their reliability. The extremely short product life cycle in consumer

electronics imposes a high innovation pressure also on medical electronic devices, which demands highly efficient confirmation processes.

Corresponding to the increase in verification and validation efforts, the amount of documentation and evidence of compliance that is necessary for approval has risen dramatically in recent years. For example, the main standard of safety in medical products, DIN EN 60601-1 (VDE 0750-1)_1996, had less than 180 pages of requirements in 1996, whereas the most recent version (DIN EN 60601-1 (VDE 0750-1)_2007 with collateral standards) has more than 1800 pages of requirements. US regulations are described in [FDA 2002], for the respective European documents see [IEC 2006] and [ISO 14971].

One of the main benefits of model-based design is that it can help to reduce the manual verification and validation efforts, since it provides a well-founded methodology with a high potential for tool support (cf. [Hungar and Reyzl 2008]). Furthermore, models can be used in the documentation of critical systems and provide a solid basis for assessment by authorities. In the following, we will demonstrate challenges and solutions using a typical example from the healthcare domain.

15.2 Evaluation Case Study: Extended Care System

In this section, we describe a prototypic *Extended Care System* (ECS) that was developed to evaluate the modeling theory of this book. Our prototype of an ECS is typical for a number of similar systems. In particular, our case study exhibits aspects of all above-mentioned current trends in the field (interconnectedness, telematics, therapy, and ambient assisted living). Furthermore, it includes a medical product of the highest safety class. Our system provides an extended range of intensive care for patients with a serious heart condition, such that they can stay in their home environment but remain under intensive care.

15.2.1 Medical Emergency Support Systems

Firstly, we characterize overall requirements and scenarios for medical emergency support systems. In general, this type of system involves three different actors: the patient, the emergency service provider, and the emergency help.

Main purpose of the ECS: quick help in case of an emergency

The main purpose of any emergency system is to provide *quick help in the case of an emergency*. In our case, an emergency can be a fall to the ground, cardiac arrest, apnea, or general immobility. The patient demands a reliable system that, on the one hand detects every emergency; on the other hand, the system must transmit only actual emergencies to the emergency service provider, since unnecessary emergency calls lead to additional efforts and deferral in the handling of actual emergencies.

The task of the emergency service provider is to handle incoming alerts. In order to do so, the data transmitted from the patient side must be comprehensive and understandable so that the provider can evaluate and prioritize the emergency correctly. Furthermore, timely delivery of the data is important to enable prompt reaction. For some systems, the emergency service provider must also be able to constantly monitor all of the patient's relevant vitality data. In medical systems, all personal data of the patient must be treated as confidential and protected against corruption and modification.

In case of an alarm, the emergency service provider may notify and consult emergency help, i.e., a designated doctor or ambulance. The physician has access to the patient's medical record and prescribes further medication or treatment. Additionally, the medical emergency support system may also give advice to the patient for activities to improve his health.

Basic scenario: quick help in the case of an emergency

Our case study is constrained to the scenario *collapse of the patient*. A possible trigger for this scenario is the patient stumbling while walking and subsequently falling down. The patient cannot stand up or crawl on his own and thus remains immobilized. The system detects the collapse when it recognizes a powerful acceleration of the body and limps, a change in the altitude of the upper body (as measured by a barometric sensor), as well as a higher heart rate due to the shock. The sensors attached to the patient's body recognize these changes and exchange their measurement results. Finally, the sensors conclude that the patient has fallen and subsequently alert the emergency service provider.

The service provider receives the emergency alarm and further information about the situation. This information comprises live data from the sensors as well as data from the last minutes before the collapse. The service provider then notifies the emergency help personnel, who take further action.

Negative scenarios

A negative scenario describes what the system must not do. We identified two negative scenarios: the first scenario is that the patient falls and is subsequently immobilized but the system does not detect the collapse (*no alert*). The second scenario is that the patient lets himself sink into his bed, but the system incorrectly detects an emergency and alerts the emergency service provider (*false alert*).

15.2.2 System Structure of the Case Study

The ECS comprises and integrates three components: a ventricular assist device (VAD), a body area network (BAN), and a telematic system (TMS).

System structure of the ECS: VAD, BAN, and TMS

In our case study, the ECS is an extension to an existing heart support system. Thus, we extended an existing VAD by a BAN and TMS. The integrated components form our extended care system for medical emergency support, depicted in Fig. 15-1.

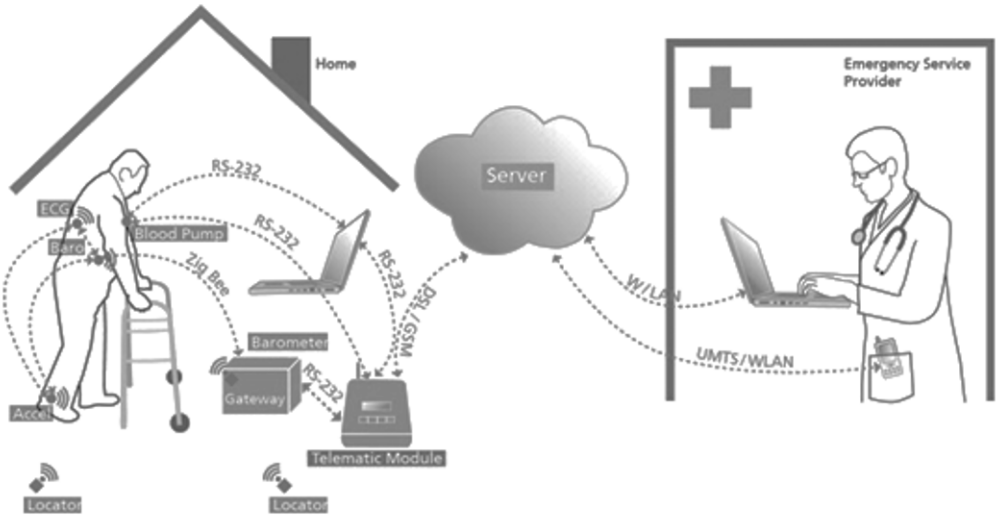


Fig. 15-1 *Extended care system*

Ventricular assist device (VAD)

The VAD supports the patient's heart by augmenting some or all of the heart's pumping capacity. It consists of a blood pump with a microcontroller and a control laptop. The laptop provides a control panel for configuring and monitoring parameters of the microcontroller and

displaying patient data and error messages. The control laptop interfaces with the pump's microcontroller by means of a special protocol designed to fulfill the compulsory safety requirements. Cyber-physical modeling of a similar system can be found in [Jiang et al. 2011].

In an ECS environment, the VAD has to be protected against (accidental or malevolent) misuse. Thus, before attaching additional components to the VAD, the following compatibility issues have to be considered:

- The transfer of data from and to the VAD has to take place via a secure channel.
- The other components may not at any time change, control, or interrupt the functionality of the VAD.
- The patient should retain sovereignty over his data; that is, except for emergency situations, a transfer of data may only take place with the patient's approval.

Only if these conditions are met can the BAN and TMS components of the ECS be used as additional components to the VAD. More remarks on high-confidence medical device software can be found in [Lee et al. 2006].

Body area network (BAN)

The BAN consists of various sensor nodes attached to the patient's body and spread around the residence. The sensor nodes form a subnet within the ECS, hence the term *body area network*. A gateway connects the BAN to the TMS and subsequently to the emergency service provider. The BAN has a decentralized structure; there is no central node collecting the data from all sensors as this would form a single point of failure. The nodes may communicate with each other if an exchange of data is necessary to pronounce a more reliable verdict.

The wireless communication among the sensor nodes and the gateway has advantages as well as disadvantages: sent packets may get lost or may be read by anyone within reach. They also have to deal with the fact that other nodes may join or leave the network at any time, expectedly or unexpectedly (e.g., due to power failure). Hence, an authentication algorithm protects the nodes against misuse and a special protocol ensures that if a node is within reach, it will receive its messages eventually.

The BAN component of the ECS is not a medical device in the legal sense, as it is not used for diagnostic purposes in a therapy or medical treatment. Although it tries to identify critical situations and summon

professional assistance, it does not actively harm the patient if it fails to do so.

Telematic system (TMS)

For remote monitoring of the VAD and the BAN, the ECS comprises a telematic system. The TMS consists of three components: a telematic module, a server, and the client software. The telematic module transfers the data from the patient side to the emergency service provider. The server hosts the business logic, data backup, and user management. The doctor's client software is for visualization of the patient's data.

The TMS has three main features: online monitoring, alert handling, and data logging. Online monitoring enables the doctor to receive real-time data from the patient side. The live data can be visualized either via a web browser or a Smartphone. Alert handling deals with incoming emergency alerts triggered by the BAN. An emergency dispatcher accepts the alert and decides, on the basis of the data sent, whether further measures have to be taken or the alert is a false positive. Data logging provides the option of transmitting a patient's data to a global data storage. The logged data can be retrieved and evaluated with special software for retrospective diagnosis by a physician.

When integrating the VAD in the ECS, the exception to the third requirement of the VAD, "sovereignty in the transmission of patient data" has to be considered. A patient who falls and possibly loses consciousness cannot and should not intervene in the process of the emergency call. For this reason, the TMS provides a service with which it can independently and autonomously build up a connection to the emergency service provider. The security of this feature must be maintained in the design of the ECS.

15.2.3 Challenges of the Case Study

The case study offers many challenges for modeling, validation, and certification. Since this book describes a specific modeling theory, we subsequently focus on the following aspects:

- Requirements specification
- Design and evaluation of system architectures
- Interface definitions in complex medical systems

Further challenges that are of interest but that are not dealt with in this chapter comprise early validation of the system's requirements, model-based security analysis, testability of requirements, and functional safety for distributed processing in life-sustaining systems.

Safety-critical (Class III) medical products

The VAD is a life-sustaining system and thus is classified as a highly safety-critical product (medical device Class III according to the council directive 93/42/EEC and software category C according to IEC 62304). Therefore, intense validation and verification is needed before regulatory authorities permit release to the market. The BAN and the telematic system are of less concern. One of the challenges of the case study is how to maintain the Class III certificate of the VAD while adding further, not necessarily Class III, components to the environment. It is essential to avoid having to certify the other parts of the ECS as Class III medical devices as well to save efforts for validation and verification. For a successful integration of the BAN and the TMS into the VAD, there must be a guarantee that the combination of the three has no impact on the functionality of the VAD. Therefore, the interfaces between all components have to be designed with great care.

15.3 Example Evaluation Activities in Detail

After describing basic functionality, system structure, and challenges for the ECS in the previous section, we now describe the application of our modeling theory to this case study. To do this, we largely rely on established UML2 or SysML diagrams; see [Raistrick et al. 2004]. In order to manage the complexity of the entire system, different concepts and techniques should be used at various levels of abstraction, as described in Chapters 4 through 7. We describe our modeling using some sample artifacts generated within the development of the ECS.

15.3.1 Requirements Elicitation

The first task in the system development is to identify the boundaries of the system to be developed.

With regard to the ECS, a question relating to the system boundaries was whether calling the emergency help and the subsequent questions of the high availability of and safe communication to the emergency help should be included in the system or be considered as a part of the environment. In the following, we only describe the second alternative.

Modeling the requirements viewpoint with block diagrams, requirements diagrams, and use case diagrams

In accordance with the requirements viewpoint (see Section 4.2.1), we used SysML block diagrams to model the system boundaries (see Fig. 15-2). As described in Section 15.2.1, we distinguish between emergency service provider and emergency help as different actors in the environment. Hence, the emergency call system has to transmit not only alarm messages, but also information on the current patient condition.

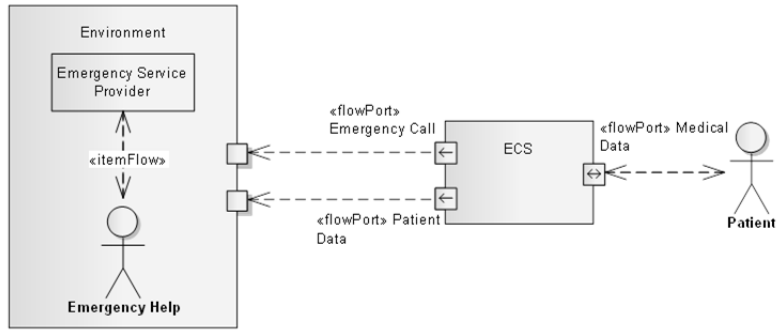


Fig. 15-2 *Context diagram for the extended care system*

According to the requirements process model described in Chapter 4.4, the next step is to define the goals the system has to fulfill. Recognizing the collapse of the patient and sending an alert to the emergency service provider is a main goal of the system. As described above, the emergency service provider and emergency help have limited resources and should be called only if they are really needed. Hence, there are two conflicting goals: calling the emergency service provider immediately in an emergency, and avoiding false alerts. We modeled the goals using SysML requirements diagrams, as suggested in Section 4.2.2 (see Fig. 15-3).

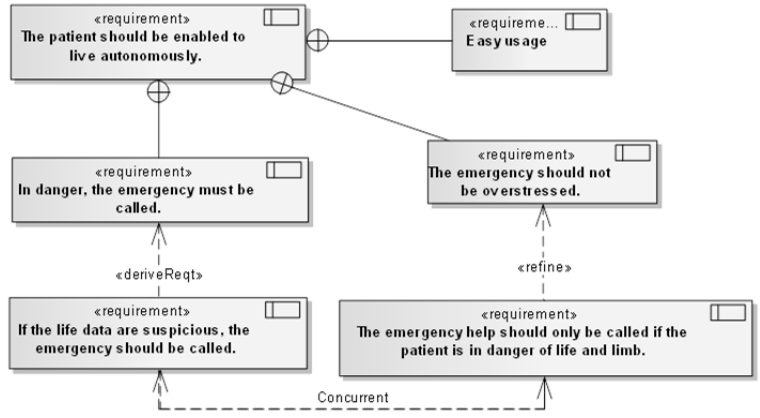


Fig. 15-3 *Concurrent goals for the extended care system modeled by SysML requirements diagrams*

At this point we need a better understanding of the interaction of the system with its environment: How should the ECS behave in a critical situation? What are possible critical use cases? These aspects can be

analyzed properly with the help of SysML sequence diagrams and structured by use case diagrams, as suggested in Section 4.2.3. Some use cases are depicted in Fig. 15-4.

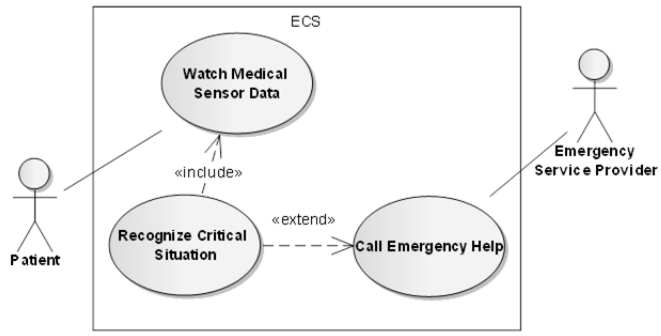


Fig. 15-4 *Use cases of the extended care system*

Sequence diagrams for modeling scenarios

The extended care system should permanently monitor the patient data and notify the emergency service provider if a critical situation occurs. The scenario *collapse of the patient* is demonstrated by the sequence diagram in Fig. 15-5 (the time axis of this diagram points down vertically).

In order to resolve the concurrent goals mentioned above, we decided that a call-back to the patient by the emergency service provider must be performed. If the patient responds to the call-back, the help request generated by the alert can be cancelled by the emergency service provider.

15.3.2 Structural Investigation of the Requirements

The requirements process model, as described in Section 4.4, prescribes that after developing solution-neutral artifacts such as the ones shown in Section 15.3.1, solution-oriented artifacts have to be developed in order to represent the solution concept. After developing the logical structure and behavior of the system, we have to transfer it to a physical realization.

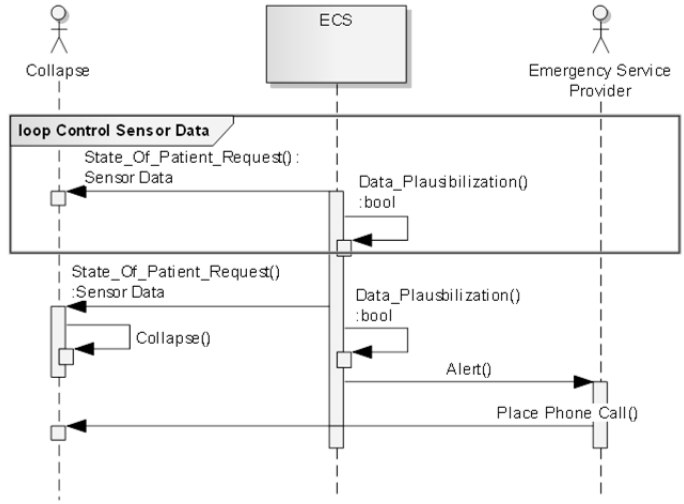


Fig. 15-5 *Collapse scenario modeled by a sequence diagram*

Modeling patient data by means of class diagrams

As identified in the context diagram in Fig. 15-2, the extended care system must provide all information necessary for immediate help. A system identifier, a unique name for identifying the calling system, and probably other personal data of the patient enable the emergency service provider to look up the patient's anamnesis data in the patient database. Additionally, data on the patient's current state gathered by the extended care system can be important for the emergency help, thus these data should be transmitted as well. To represent these data, we model them with SysML class diagrams, as suggested in Section 4.2.4. The model is shown in Fig. 15-6.

15.3.3 Functional Decomposition

Once the requirements have been set, the functional structure of the system as a black box, i.e., from an external point of view, has to be modeled according to Chapter 6.

Deployment diagrams for modeling the functional viewpoint

We used UML deployment diagrams to model the functional architecture of the (BAN part of the) ECS (see Fig. 15-7). These diagrams offer hierarchical structuring and can capture the functional decomposition, similar to the graphical notation suggested in Chapter 6. For this case study, the main function of the ECS is to alert the emergency service provider in case of an emergency. An additional function is to send available diagnostic information. In order to detect a collapse, the system must monitor various physical sensors. In order to

send diagnostic data, it must access the data logging storage etc. In this way we described the functional behavior of the ECS in guiding the deployment of the system onto its components.

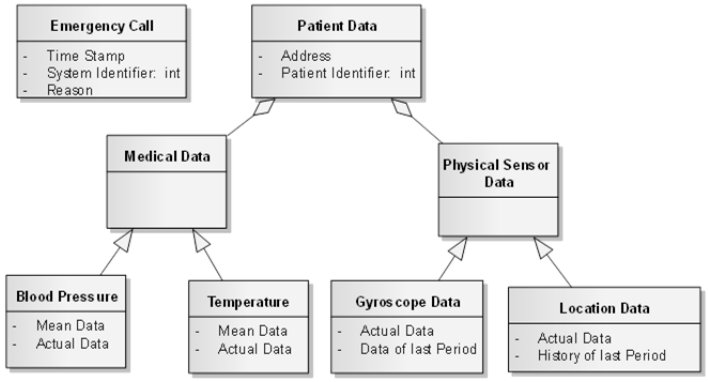


Fig. 15-6 *Information to be sent to the emergency service provider modeled by a class diagram*

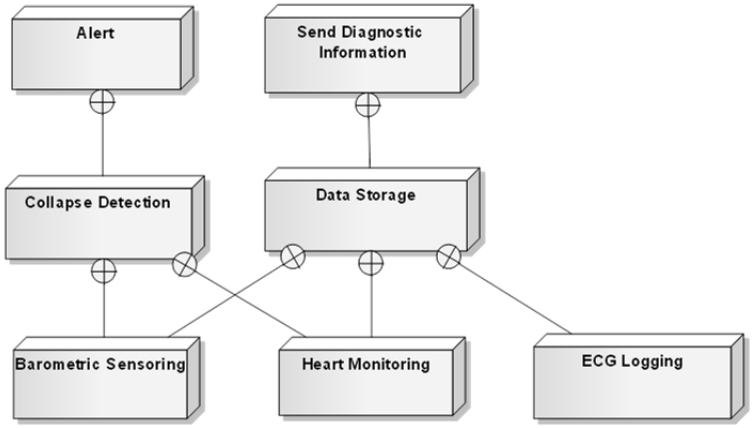


Fig. 15-7 *Functional decomposition of the extended care system using deployment diagrams*

15.3.4 Towards System Design

The previous subsections assumed a black box view of the system to be developed. That is, we looked from outside onto the system. This provides a good understanding of its behavior as expected by the user. In the next step, the implementation is conceived by designing the architecture, based on the logical and technical viewpoints (Chapters 6

and 7). Starting from the functional decomposition of the system, a proper internal structure, as well as a structure for hardware and software resources, is defined using appropriate modeling techniques.

Package diagrams model the technical viewpoint

By considering the BAN from a technical viewpoint, we decided to incorporate mobile and stationary nodes in order to meet the goals "easy usage" and "monitoring of various sensor data." Moreover, the TMS, responsible for the communication with the emergency service provider, is included in the technical structure. It is connected to the sensor nodes via a gateway. In this context, the VAD can be thought of as just another sensor providing information about the cardiac activity of the patient. We modeled the architecture using SysML package diagrams, as shown in Fig. 15-8. The various functions as identified in the functional decomposition model of Fig. 15-7 were then mapped to the nodes that should realize them.

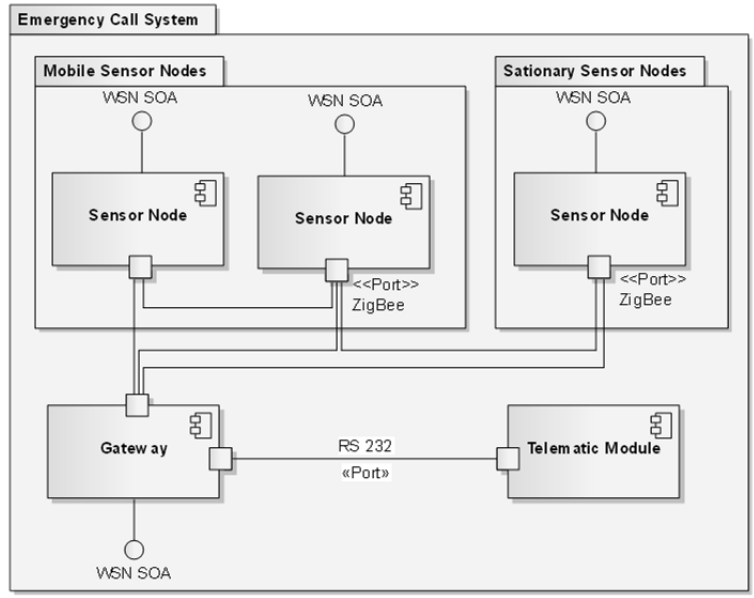


Fig. 15-8 *Architecture of the extended care system using package diagrams*

Integrating legacy code via capsules

For the VAD, most implementation parts already existed. In order to incorporate it into the case study and to enable certification of the complete product, we also designed models for the VAD. Such models are helpful for demonstrating the functional integrity of the system. The general problem is how to integrate legacy code in a model-based development environment. As a solution, we used a reengineering

approach. We developed a capsuling concept, where the interfaces between a capsule and its environment were derived from an integrated analysis of code and model. Then, actors were extracted from the tasks in the code. An extended static analysis allowed us to trace the control variables of the actors to derive activity diagrams and state charts. With these generated models, we were able to guarantee that BAN and TMS had no negative effects on the workings of the life-support system.

15.4 Summary

In modeling the various parts of the case studies, we used several modeling notations as defined by the SPES modeling framework. The flexibility of the formalisms supplied was helpful in engineering the models. One challenge turned out to be the still nonexistent tool integration, at syntactic as well as at semantic level. Models that are developed with one tool (e.g., an architecture modeler) cannot necessarily be re-used with other development tools (e.g., test generators), see [Tahirbegovic and Lackner 2011].

Accreditation based on modeling artifacts

Our ECS case study comprised several components, with the patient equipment built on top of an existing VAD system. For such highly safety-critical systems, accreditation by notified bodies is of utmost importance. As mentioned above, the noninterference of the environment with the workings of the VAD must be shown. Here, models and diagrams can be extremely helpful, see also Chapter 9 on modeling safety aspects. Our experiments also showed the importance of modeling a system's context, as well as unwanted and disallowed use cases. With a large number of models, however, it becomes increasingly difficult to maintain the consistency of the various models and artifacts. In particular, showing that the actual implementation conforms to the different models can only be done partially. Automated procedures such as model checking are helpful only to a certain extent, since many modeling concepts are beyond the scope of a model checker. SysML offers certain constructs that can be used to mitigate this problem. Checking interdependencies between the models still remains challenging.

The increased usage of software in safety-critical medical systems allows a high flexibility for improving and extending a given functionality. In order to be able to use this flexibility and at the same time meet the high safety standards, concepts for modular validation and re-certification are needed. Ideally, validation and verification should start in early development phases, allowing reuse of models, component

descriptions, and code fragments together with their respective validation documents in an incremental development process.

15.5 References

[FDA 2002] Food and Drug Administration: General Principles of Software Validation. San Francisco: U.S. Department of Health and Human Services, 2002.

[Glesner et al. 2007] S. Glesner, S. Jähnichen, B. Paech, B. Rumpe, T. Wetter, A. Winter: Strategische Bedeutung des Software Engineering für die Medizin. In: W.-G. Bleek, J. Raasch, H. Züllighofen (Hrsg): Software Engineering 2007, Fachtagung des GI-Fachbereichs Softwaretechnik. Springer LNI P-105, Hamburg, 2007, pp. 25-28.

[Hungar and Reyzl 2008] H. Hungar, E. Reyzl: Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme: Bedeutung modellbasierter und formaler Ansätze für effiziente Entwicklung und Zertifizierung. In: Proceedings of Software Engineering 2008, pp. 291-294.

[IEC 2006] International Electrotechnical Commission: IEC 62304:2006(E): Medical device software – software life cycle processes. First edition, May 2006.

[ISO 14971] ISO International Organization for Standardization: 14971: Medical devices – Application of risk management to medical devices, 2000.

[Jiang et al. 2011] Z. Jiang, M. Pajic, R. Mangharam: Cyber–physical modeling of implantable cardiac medical devices. In: Proceedings of the IEEE, Vol. 100, No. 1, 2011, pp. 122-137.

[Lee et al. 2006] I. Lee, G. J. Pappas, R. Cleaveland, J. Hatcliff, B. H. Krogh, P. Lee, H. Rubin, L. Sha: High-confidence medical device software and systems. In: IEEE Computer, Vol. 39, No. 4, 2006, pp. 33-38.

[Raistrick et al. 2004] C. Raistrick, P. Francis, J. Wright, C. Carter, I. Wilkie: Model driven architecture with executable UML. Cambridge University Press, Cambridge, 2004.

[Tahirbegovic and Lackner 2011] S. Tahirbegovic, H. Lackner: Systematischer Test für vernetzte Medizingeräte: Nicht kapitulieren, sondern automatisieren. http://epaper.konradin-relations.de/medizin+technik/iframe/2011005/. Accessed on: April 3, 2012.

[Zauner and Schrempf 2009] M. Zauner, A. Schrempf: Informatik in der Medizintechnik: Grundlagen, Sichere Software, Computergestützte Systeme. Springer, Vienna, 2009.

Dr. Andreas Jedlitschka
Jessica Jung
Dr. Constanza Lampasona

16

Evaluation Summary

A survey conducted by the developers of the SPES modeling framework at the beginning of the project revealed four high-level industry challenges that were to be addressed. Several evaluation studies were conducted in the application domains, aiming at investigating the contributions of the SPES modeling framework towards solving these industry challenges. The results of these studies are summarized for each challenge. The main results indicate that the SPES modeling framework is applicable in the chosen application domains. However, additional efforts for adapting the SPES modeling framework were reported to be necessary. In summary, the SPES modeling framework is well-aligned with regard to the industry challenges. Future evaluations would be necessary to investigate cost benefits and the efficiency of the approach. In addition, evaluation results show that the SPES modeling framework has the potential to provide a stable foundation for future evolutions of model-based approaches for the development of embedded systems. The current need and future relevance of model-based development approaches for embedded systems is supported by results from a survey.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_16,

16.1 Introduction

Motivation: empirical studies as decision support

Decision makers in embedded system development require evidence about technologies to enable them to make an informed decision when new development technologies are to be introduced. However, this evidence must be obtained systematically, i.e., the studies have to be planned according to goals, conducted, analyzed, and reported so that the results are conclusive and the studies can be replicated. In Chapter 10, we presented an overview of the evaluation strategy that was to be used to evaluate the SPES modeling framework.

Matching the SPES principles to the SPES modeling framework

Industry challenges for the engineering of software-intensive embedded systems were identified by means of a set of interviews conducted with project partners, as well as a comprehensive state-of-practice study (see Chapter 2, [Sikora et al. 2012]).

We summarized the given industry challenges, identifying four high-level challenges:

- Model technical systems and their interaction
- Provide traceable and seamless support for all life cycle phases
- Address and verify system properties early
- Address safety, standard compliance, and certifiability

Based on these challenges, principles for the SPES modeling framework were derived (see Chapter 3):

- Distinction between problem and solution
- Distinction between logical and technical solution
- Explicit consideration of system decomposition
- Seamless model-based engineering
- Continuous engineering of crosscutting system properties

The SPES principles have to be addressed by the SPES modeling framework. For the SPES modeling framework, the following properties were defined:

- Scalability
- Productivity
- Division of labor
- Ease of use

For the SPES models the following properties were defined, describing the extent to which they support system development:

- Handling of complexity
- Maintainability
- Reusability
- Consistency between different models (in all directions of the SPES modeling framework)
- Reduction in fault tendency

The high-level evaluation goals addressed in the evaluations in the application domains were quality, efficiency, and usability.

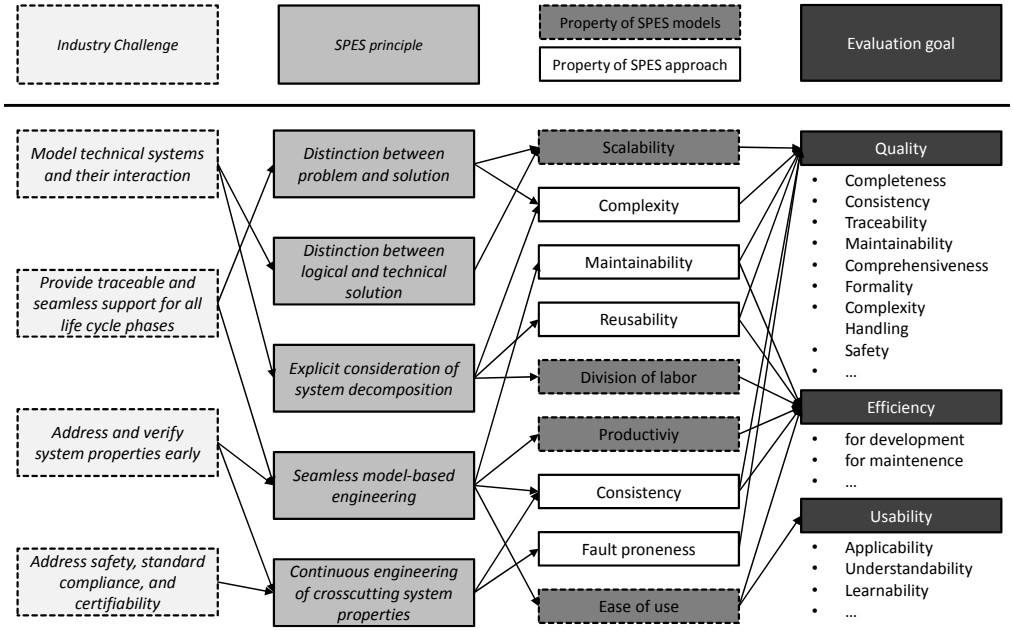


Fig. 16-1 *Matching SPES principles to the properties of SPES modeling framework and to the evaluation goals*

Fig. 16-1 presents the relationships between the summarized industry challenges, SPES principles, properties of the SPES modeling framework and inherent models, and evaluation goals. The evaluation in the application domains assessed the SPES modeling framework with regard to specific attributes listed as evaluation goals. For example, with regard to the evaluation goal quality, attributes such as a model's completeness, its consistency, and traceability of artifacts were studied. Also, the question of the extent to which the SPES modeling framework had an

impact on the development or maintenance efficiency was investigated. In terms of the evaluation goal usability, the SPES modeling framework was evaluated concerning aspects such as applicability, understandability, and learnability. A selection of evaluation studies were reported in previous chapters.

In the remainder of this chapter, we summarize the results from the studies that were conducted to evaluate the SPES principles and their implementation in the SPES modeling framework. Further, we discuss results from a survey that support current and future relevance of model-driven development (MDD).

16.2 Conclusions from the Evaluations

Overview and conclusions of evaluation studies in SPES

To assess the extent to which the SPES modeling framework and models fulfill industrial challenges, the evaluation studies in the application domains addressed both the level of abstraction and the viewpoints. Fig. 16-2 shows that the evaluation studies cover the whole SPES modeling framework.

A vertical bar indicates that for an empirical evaluation of an approach, the approach was applied across several abstraction layers within a specific viewpoint, for example, the requirements viewpoint. A horizontal bar indicates that the focus of the study was across viewpoints. For the details of the evaluations we refer to Chapters 11 through 15.

The general purpose of the evaluation studies was to investigate whether and under what conditions the SPES modeling framework can be applied in the context of the specific application domain (cf. domain-specific challenges). Domain-specific requirements have influenced both the adoption of the SPES modeling framework as well as the design of the evaluation studies. These domain-specific requirements come from:

- Disciplines with heterogeneous engineering approaches, such as mechanics, electrics, and software, that need to interact in the automation (cf. Chapter 11) and automotive domains (cf. Chapter 12)
- Disciplines with the need for certifiable systems in the avionics (cf. Chapter 13) and healthcare domains (cf. Chapter 15)
- Disciplines with the need for flexible handling of massively distributed systems in the energy domain (cf. Chapter 14).

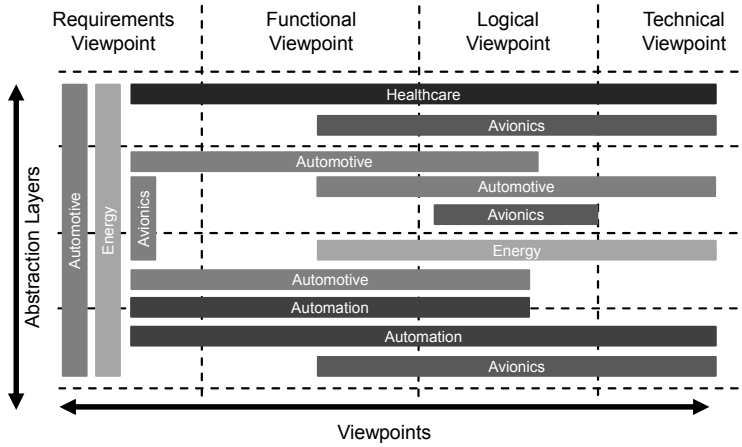


Fig. 16-2 *Placing the evaluation studies within the SPES modeling framework*

Summary of results of the evaluation studies

Most studies addressed the question of whether the SPES modeling framework can be applied (applicability) in the context of the specific domain. All domains report that the SPES modeling framework and inherent models are applicable. Nevertheless, a need for adaptation and a detailed guideline was reported. Some possible adaptations have been exemplified, e.g., in Section 14.4.1. Learnability of the technologies and understandability of the resulting documents was also perceived as positive by participants. However, in a few cases, explicit improvement suggestions, in particular regarding the transfer into daily practice, were given (e.g., see Section 12.3).

In the following, we summarize the results of the evaluation studies from the perspective of the high-level industry challenges.

Model technical systems and their interaction: Each of the five application domains applied the SPES modeling framework to several (domain-specific and typical) case studies. The case studies successfully demonstrated that the SPES modeling framework addresses this challenge. For example, in the automation domain, several types of systems were successfully integrated by mapping SPES principles to domain-specific modeling languages. In the avionics domain, we successfully showed that systems engineering and safety engineering can be integrated more smoothly by employing the SPES modeling framework. As can be concluded from Fig. 16-2, much emphasis was given to the requirements viewpoint. This is quite obvious, because this viewpoint addresses also understanding of the system to be developed and a common language for stakeholders involved. Results from the case

studies demonstrate that the SPES modeling framework supports system understanding.

Provide traceable and seamless support for all life cycle phases: The challenge of horizontal integration (i.e., seamless methodological and tool support) of the SPES modeling framework (i.e., across several viewpoints) was addressed in all domains. As shown in Fig. 16-2, all domains provide case study results demonstrating how they employed the SPES modeling framework. Although the general results provide supporting evidence, the level of integration is different. Whereas for the automotive domain, seamless integration was successfully demonstrated for approach and development tools, the healthcare domain recognizes a deficiency with regard to supporting tools (see Chapters 12 and 15, respectively). Results reported from the case study in the avionics domain provide the insight that the integrated design and safety modeling allows systems and software engineers to work seamlessly on the same model (see Chapter 13). In addition, the results show that parts of the verification cases can be generated from formalized requirements.

Address and verify system properties early: Several studies investigated aspects of quality (cf. Fig. 16-1). With regard to the question of whether a certain level of product quality is achieved using the SPES modeling framework, we can summarize that the methods specifically addressing a certain quality aspect such as completeness, consistency, safety, or traceability fulfilled users' expectations. In particular, approaches addressing requirements' vertical traceability were successfully investigated, e.g., in the automotive, avionics, and energy domains (see Chapters 12 through 14 respectively). Results from a case study on an extended requirements process model in the automotive domain show a shift of effort towards earlier phases. Similarly, system simulation as a means for early verification was successfully integrated and employed in cases studies, in particular in the automotive and energy domain.

Address safety, standard compliance, and certifiability: Within the automotive, avionics, and healthcare domains in particular (cf. Chapters 12 through 14), the topics of safety, standard compliance, and certifiability are highly relevant. Therefore, these topics were also addressed in the case studies, e.g., a case study in the automotive domain demonstrated that the logical architecture can be automatically transformed into AUTOSAR application components and can be transferred to AUTOSAR basic software configurations.

In the avionics domain, it was found that the concept for the integration of safety cases as argumentation support for the certification

authority showed a potential towards automated certification of safety-critical systems.

The integrated design and safety modeling showed that systems and software engineers can work seamlessly on the same model.

Transferability of the results across domains: As can be seen from previous chapters, different aspects of the SPES modeling framework were evaluated in different domains. For example, the automotive domain – among other things – focused on functional correctness and completeness in early phases as well as traceability (see Chapter 12), while the avionics domain focused on safety and verification (see Chapter 13). The results indicate that the transfer across all domains is possible in principle, but required adaptations according to the characteristics and challenges of the specific domains, e.g., in the automation domain to cope with different modeling languages of engineering disciplines involved, in the automotive domain to manage variants, and in the avionics domain to address safety and certification.

Open topics and future work: To provide decision-makers with even better guidance regarding the benefits or shortcomings of the SPES modeling framework, additional information, in particular, business-relevant aspects such as the costs and efficiency of methods, should be gathered by systematic empirical evaluation. However, this would require access to historical project data, serving as a baseline, and a set of real projects in which the SPES modeling framework is applied. The results of these projects would then be compared against the baseline. The data is hard to obtain due to confidentiality constraints. Furthermore, access to real projects often proved to be difficult due to the scheduling and alignment restrictions of those projects.

16.3 Relevance of Model-Driven Development

Survey results: major benefits of MDD for both industry and academia

To investigate the relevance of model-driven development (MDD), a survey was conducted [Lampasona 2012]. The questionnaire consisted of 87 items and was divided into six parts, including demographics, importance of MDD today, importance of MDD in the future, and expectations of MDD. The survey was designed taking into account two specific groups: participants from industry (both development staff as well as executive positions were included) and from academia, and 127 people were invited to participate. Of those, 64 answered the questionnaire.

The results show that MDD is more important in contemporary software engineering research than for current business objectives. For

managerial staff, on average, MDD is equally important in both research and business. However, a vast majority agreed that the future of software engineering lies in MDD. In this case, half of the business executives strongly agreed, others agreeing mostly or answering neutrally on this issue.

Furthermore, we asked participants to judge the extent to which they expect different properties from MDD. As can be seen from Fig. 16-3, improving quality and reuse, complexity management, and reduced development costs are the issues that both researchers as well as practitioners expect the most benefit from by using MDD.

Finally, we asked an open question about the expectations of participants with regard to MDD. Out of 185 responses, the most frequently mentioned points were: improved validation and verification (19 responses), shorter development times (19), increased automation (15), better quality (16), and reduced costs (11).

In summary, we can say that model-based development is not only economically relevant, but is also expected to introduce major benefits for both industry and academia. In the SPES 2020 project, we have laid the foundation for successful model-based engineering of embedded systems, but there is still much to be done.

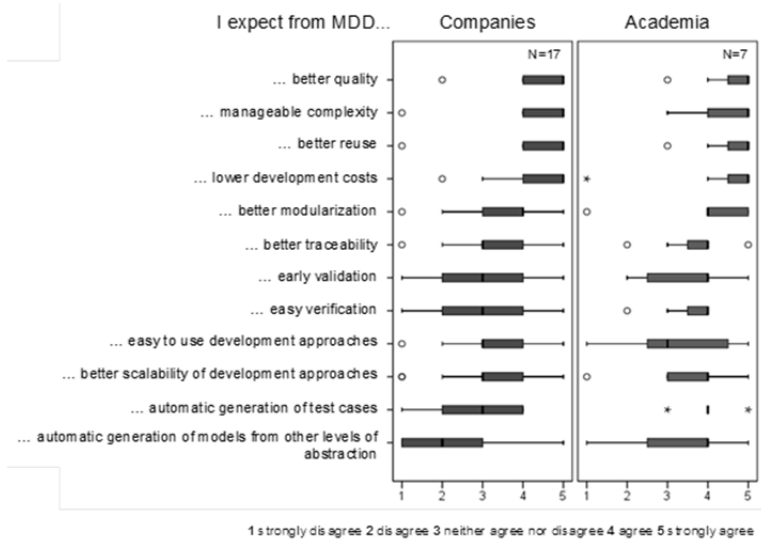


Fig. 16-3 *Expectations from MDD of practitioners and academics*

16.4 Summary

Positive results from individual evaluations: SPES approach and adaptable and applicable models

The predominantly positive results from the individual evaluations indicate that the SPES modeling framework and its inherent models address the industry challenges. Furthermore, the case studies in the application domains showed that the SPES modeling framework and models are adaptable to and applicable within the application domains. The SPES modeling framework, with its abstraction layers and viewpoints, helped to close existing gaps in systems and software modeling practice by providing, e.g., a better understanding of how to refine and to decompose complex systems and how to describe the relationship between artifacts created. These results are underlined by results from a survey that show that MDD is becoming more and more relevant. However, questions regarding the approach's cost-efficiency and impact on the development schedule have to be answered by future empirical studies.

16.5 References

[Lampasona 2012] C. Lampasona: Umfrage SPES 2020: Relevanz, Zukunft und Stellenwert modellbasierter Softwareentwicklung. Project Deliverable of SPES 2020. Available at spes2020.informatik.tu-muenchen.de/results/ZP-AP6.D6.2.B1 MDD Umfrage_final.pdf

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal, Vol. 17, No.1, 2012, pp. 57-78.

16.6 Acknowledgements

We specifically acknowledge the contributions of Marcus Ciolkowski, who led the evaluation work in SPES 2020 until the end of May 2011. Special thanks go also to Sabine Nunnenmacher for her contributions in planning and conducting empirical studies and surveys in the first two years of SPES 2020. Last but not least, we also want to thank Sarah Tichy for her support in the statistical analyses of data obtained from the studies conducted, and Bastian Tenbergen for valuable comments and never-ending patience.

Part IV

Impact of the SPES Modeling Framework

Peter Heidl
Jens Höfflinger
Harald Hönninger
Bastian Tenbergen

17

Lessons Learned

The purpose of this chapter is to offer a view on the lessons learned from developing and applying the SPES modeling framework. The lessons learned show that the SPES modeling framework is well-aligned with regard to the industry challenges. In addition, results show that the SPES modeling framework has the potential to provide a stable foundation for the further evolution of model-based approaches for embedded system development.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_17,

As proposed in Chapter 2, the engineering of modern embedded systems is becoming increasingly challenging due to a steady increase in demand for more innovations in shorter time to market as well as overall cost pressure. Current development approaches for the engineering of software-intensive embedded systems are ill-equipped to meet these high demands. Consequently, novel development paradigms are necessary that meet individual demands of the embedded systems world. Therefore, we developed and evaluated the SPES modeling framework as a potential solution to the challenges arising in embedded system development. The project was a success, as challenges such as complexity management, which was previously thought of as an insurmountable obstacle, is now one of the core SPES principles and has become an indispensable tool for profitable engineering of embedded systems.

Lessons learned in developing the SPES modeling framework

The SPES modeling framework is a milestone for the model-based development of embedded software in the industry. The most striking factors that contributed to the success of the SPES modeling framework can be summarized as the following lessons learned:

- The multitude of systems gives rise to a multitude of engineering challenges.
- Model-based software development is increasingly important.
- Integrated development is essential for the engineering of embedded systems.
- Interdisciplinary knowledge networks foster innovation.
- We have achieved a lot — but a lot more still remains to be achieved.

In the following sections, we will give a short discussion of each lesson learned.

17.1 The multitude of systems gives rise to a multitude of engineering challenges

Not all embedded systems are created equal

The term *embedded system* appears to describe a relatively clear-cut type of system. However, on closer inspection, it becomes obvious that embedded systems are more likely to be a class of systems that comprise technical as well as nontechnical aspects, such as business model, product properties, problem classes they address, context conditions they may encounter, etc. In other words, the term *embedded system* may describe a system that has multiple systems embedded within itself, such as a cockpit in the avionics domain or a rolling mill in the automation domain. On the other hand, the term *embedded system* may also refer to a

system that is embedded within an environment of other systems, such as an engine control unit in the automotive domain.

In addition, dominant product properties may vary in aspects such as criticality, safety, variability, real-time constraints, etc. The problem classes embedded systems are meant to address may include continuous closed-loop control tasks, data-centric computing problems, or user-centered interactions.

These aspects lead to different processes in the engineering of such systems and result in vastly different architectures and solution approaches. However, there are also some commonalities that may be regarded as conceptual building blocks for specialized development concepts. Some of these building blocks have been translated into SPES principles and were addressed in Part II and Part III of this book.

17.2 Model-based software development is increasingly important

Increasing volume of embedded systems drives need for cost-efficiency

Embedded systems development is a lucrative business with total revenues ranging to billions of euros. The number of embedded systems in the world is steadily increasing as these systems have become an integral part of our daily lives. For example, vehicle systems that were previously entirely mechanical, such as the braking system, are more and more often being implemented by electronic means using embedded systems [Volpato 2004]. Furthermore, due to the increasing interoperability of functions and features, system complexity is steadily increasing.

In order to remain profitable, development of such systems must be cost-efficient, deal with increasing system complexity and increasing quality demands, and do so in short time to market. However, when faced with the challenges of modern embedded system development, traditional development methods are lacking with regard to these aspects. In particular, meeting high quality demands is impaired by the increased product complexity.

Therefore, novel development paradigms must be established in the industry. One such paradigm is model-based software development: it promises increased productivity and therefore faster time to market, increased quality due to constructive consistency, and improved manageability of system complexity through abstraction. It is therefore reasonable to conclude that model-based software development is a core technology for the development of embedded systems.

Model-driven development in industry and academia

As part of the SPES 2020 project, a survey was conducted (see Section 16.3) to characterize the significance of model-driven development (MDD). The results indicate that model-based development is not only economically relevant, but is also expected to introduce major benefits for both industry and academia. In the SPES 2020 project, we have laid the foundation for successful model-based engineering of embedded systems; however, there is still much to be done.

17.3 Integrated development is essential for the engineering of embedded systems

Model-based development promises many benefits for the engineering of embedded systems. However, continuous development by means of integration of various engineering activities still remains essential. This means that artifacts must be continuously elicited, documented, modified and refined, starting during requirements engineering, via the modeling of system functions to their final deployment on the embedded system, across many layers of abstraction from system to subsystem to component. In addition, these development artifacts must be shared with other facets of development, such as safety and security engineering, mechanical engineering, etc.

Development continuity is necessary

The SPES modeling framework provides a basis for such development continuity, as described in Chapter 3. Initial evaluations show promising results (see Chapters 11 through 16), but also show that tailoring of the SPES modeling framework is necessary in almost every development context. Therefore, more evaluations and additional case studies must be conducted in order to give a better picture of when tailoring is necessary and how the engineering process can be guided such that the utmost benefit of model-based engineering can be gained.

17.4 Interdisciplinary knowledge networks foster innovation

Different domains produce different types of systems and requirements

As shown in Section 17.1, a multitude of different types of embedded systems causes new and intriguing challenges for development and fascinating avenues for research. These challenges and research avenues cannot be tackled in solitude. Instead, interdisciplinary networks of partners that ordinarily compete on the international market within and across application domains allow sharing of different perspectives and insights into the state of practice in engineering of embedded systems. Working in close cooperation with research institutes across Germany

and the world fosters innovation in research, quick knowledge transfer from research to industry, and enables evaluation of research results in industrial settings.

17.5 We have achieved a lot — but a lot more still remains to be achieved

Future work

In spite of the significant innovations of the SPES 2020 project, there are a number of areas that remain to be addressed. Some of these can be summarized as follows:

- *Engineering process prerequisites:* We have seen that tailoring of the SPES modeling framework is necessary in most development contexts. As tailoring was mostly driven by the individual properties of the respective application domain, the SPES modeling framework has been tailored in quite different ways (cf. the tailoring in the automotive domain vs. the tailoring in the energy domain). Therefore, a systematic investigation of different development contexts and engineering processes must be conducted to gain insight into what prerequisites must exist in a development scenario in order for the SPES modeling framework to be applicable.
- *Variability, deployment, reuse, and other qualities:* While functional aspects and real-time and safety concerns have been considered in the SPES modeling framework, many more system qualities remain. For example, variability has not been investigated during the development of the SPES modeling framework, yet plays a major role in the engineering of embedded systems (as can be seen, in part, in Chapter 12.3.3). Therefore, additional research should focus on extending the SPES modeling framework so that these quality aspects are considered as well.
- *Engineering artifact quality:* While the SPES modeling framework allows for the development of engineering artifacts that can be used in coordination with different development activities (such as behavior models in the requirements viewpoint and the functional viewpoint), some issues remain with regard to quality assurance. How can engineering artifacts be validated as early in the development process as possible? In particular, validation must also be done with regard to quality aspects, such as real-time, safety, variability, or deployment.
- *Modular safety assurance:* As explained throughout this book, many application domains are governed by strict safety and security guidelines and standards. Although the SPES modeling framework

aims at bridging the gap between safety and security engineering and other engineering activities, one open question is how to constructively ensure, during engineering, that important standards and guidelines are fulfilled. Also, since many development projects extend existing systems by introducing some new feature or altering the existing system, further research must be dedicated to reducing the re-certification overhead, e.g., by certifying parts of the system and only having to re-certify those parts that have been modified.

17.6 Summary

The innovation alliance SPES 2020 has laid a solid foundation for the engineering of software-intensive embedded systems. An SPES modeling framework has been developed that is based on five principles that meet the requirements of the application domains as established in Chapter 2. The SPES modeling framework has been evaluated in the different application domains. Results show that while the SPES modeling framework meets the basic needs of industrial development, tailoring is necessary in most development contexts. Also, there are a number of research areas that have not been addressed by the SPES 2020 project, yet are important for the engineering of embedded systems: How can qualities such as variability, modular safety assurance, or optimal deployment be considered during system development? How can engineering artifacts be validated as early as possible in the development process? What prerequisites must a development process fulfill in order to be able to apply the SPES modeling framework with minimal tailoring? These are all questions that future work must address.

During the SPES 2020 project, we learned that these research questions must be answered in a collaborative way by making use of multiple perspectives from both academia and industry representatives from various domains. Only thus can research take account of the wide variety of different types of embedded systems, their roles, responsibilities, and the application contexts for which they are designed. Future work must also be spent on continuing the development of continuous, model-based engineering approaches, as these are promising approaches to surmounting the obstacles posed by steadily increasing cost pressure, increasing complexity, and the demand for high product quality in the engineering of embedded systems.

17.7 References

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal, Vol. 17, No.1, 2012, pp. 57-78.

[Volpato 2004] G. Volpato: The OEM-FTS relationship in automotive industry. In: International Journal of Automotive Technology and Management, Vol. 4 No. 2-3/2004, 2004, pp- 166-197.

Prof. Dr. Manfred Broy

18

Outlook

This chapter summarizes the project and briefly outlines the project contributions. In addition, it provides insights into open challenges in the engineering of software-intensive embedded systems that have been triggered by the efforts undertaken in the SPES 2020 project. The chapter outlines the impact of these challenges on future research.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_18,

A major goal of the Federal Ministry of Education and Research (BMBF) is to support specific research endeavors in which academia and industry join forces to address challenges in the field and to provide a solid foundation for the engineering of software-intensive embedded systems in the future. The BMBF thereby ensures that Germany remains a significant location for high-tech industries. The SPES 2020 project has substantially contributed to achieving this goal.

SPES 2020 has established a fundamental modeling approach and has investigated in a number of issues comprising key concepts for advanced modeling of embedded systems. SPES 2020 has created a seamless development framework for the model-based engineering of embedded systems — this framework integrates and consolidates different existing approaches. The SPES 2020 modeling framework is defined based on three core concepts:

- *Viewpoints:* The SPES 2020 modeling framework distinguishes between four viewpoints: the requirements viewpoint defines the concepts and techniques for the systematic elicitation and specification of requirements; the functional viewpoint defines the concepts and techniques required to specify and model the system functions and their relationships; the logical viewpoint defines concepts and techniques required to decompose the system function into a system architecture of logical components; the technical viewpoint defines the concepts and techniques required to detail the logical architecture into a physical architecture that, amongst other things, specifies the hardware components of the system and the deployment of the software on those components.
- *Abstraction layers:* The SPES 2020 modeling framework explicitly defines abstraction layers to facilitate the definition of the embedded system at different levels of granularity. The concrete abstraction layers chosen for a particular system depend, amongst other things, on the application domain.
- *Seamless modeling of crosscutting properties*: In addition to the viewpoints and abstraction layers, the SPES 2020 modeling framework defines concepts and techniques for the seamless modeling of crosscutting system properties such as safety or real-time behavior.

To validate the framework, SPES 2020 has conducted extensive evaluation activities by means of case studies and experiments in five application domains. The evaluation clearly indicates that the SPES modeling framework is applicable in a wide variety of application domains and development settings, substantially supports the interplay of

different engineering disciplines such as software development and mechatronics, and is well suited for the systematic engineering of complex safety-critical embedded systems.

In fact, by developing the SPES 2020 modeling framework and evaluating it in five diverse application domains (automation, automotive, avionics, energy, and healthcare), the SPES 2020 project has delivered an elaborated methodology that supports the systematic, integrated, and seamless engineering and operation of software-intensive embedded systems. Thus, SPES 2020 has established a firm basis for the model-based development of embedded systems.

Nevertheless, there are still open challenges in the area of engineering and operating embedded systems that go far beyond the work and scope of SPES 2020. These challenges pertain to the current state of practice and are triggered by a number of key requirements in the field. Examples are:

- *Long-term system evolution:* SPES 2020 was very much focused on model-based forward engineering of embedded systems. A key challenge of embedded systems today is that they are in operation and under further development for a long period of time, frequently spanning decades. Managing the evolution of such systems is thus an essential issue, as changes in the context of the system during its operation must be anticipated and considered systematically during the system lifetime. The SPES modeling framework already ensures a systematic consideration of the system's context and thus also supports system evolution. However, in order to support the evolution of embedded systems adequately, the SPES 2020 modeling framework has to be extended by concepts for defining context adaptability and context sensitivity. Another open issue is the adequate support for the step-by-step migration from today's legacy processes for the development of embedded systems to a systematic, model-based long-term software and system evolution process.
- *Variability management:* In many cases, individual systems or networks of systems are developed that comprise a large set of similar functionality and that share similar architectures and implementations. Providing a clear separation between common and system-specific parts in the engineering of embedded systems will leverage a large potential for saving development costs and time, as well as increasing quality. It is quite obvious that modeling techniques used in product line engineering are very well suited to supporting the engineering of product and system families. However, the increasing complexity of embedded systems and networks of systems poses additional challenges for managing the variability of

such systems. Open issues include the integration of variation points into a comprehensive system modeling framework and the resource-efficient management of different variants of embedded systems, components, and networks of systems.

- *Cyber-physical systems:* Today and more so in the future, embedded systems will form networks of interacting elements that feature a tight combination and coordination of the system's computational and physical parts, i.e., a tight relation between the digital and physical worlds. One open question is how to capture the nature and the interaction between the physical systems' context and the digital nature of embedded software systems in the development process. Another big challenge is the question of how to put embedded systems into a manageable relationship to global networks such as the Internet. In the past, embedded systems were typically closed systems with a static architecture, statically fixed sets of functions, and clearly defined static interfaces to their context. When embedded systems are connected to the Internet, those characteristics change. This creates various research challenges such as the challenge of dynamic system models, hybrid system models, dynamic interface models, and dynamic models of the architecture that address these different characteristics of systems and system parts, together with certain quality requirements such as safety and security. While SPES 2020 laid a foundation for addressing these challenges, further research is required to extend the SPES modeling framework with concepts and techniques that provide solutions for these challenges.

Overall, many scientific and practical engineering challenges remain to be solved. Hence, significant research effort is still required. Moreover, dedicated effort is required to ensure that the techniques and methods developed can be easily deployed to and adopted by industries. In order to support industrial uptake, process descriptions and guidance for system engineers that can be easily adapted to the specific requirements of the individual development processes have to be developed.

In addition, future research for embedded systems has to be strongly related to the sociological and to the economic contexts of embedded systems, as most technical devices will be mutually connected, enhanced by embedded software, and connected to the cloud. Issues of man-machine interaction and advanced assistance will become dominant. Systems will be ubiquitous, pervasive, and globally connected. Only if the engineers are able to construct such systems with a proven correctness, high reliability, and usability such systems will positively enhance our reality, and make our life easier, richer, safer, and more secure.

Appendices

A – Glossary of the SPES Modeling Framework

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9,

A

Abstraction layer
An abstraction layer defines a specific level of abstraction and granularity at which the System under Development (SUD(↑)) is examined. The level of granularity of the respective abstraction layer is in turn determined by a structural characteristic that stems from the layer above. Initially, we consider the system as a whole.

D

Decomposition
Decomposition denotes the partitioning of an analysis element or design element (e.g., of a goal, a function(↑), or a logical/technical component(↑)) into parts.

Diagram
A diagram is a graphical representation of a model or of a part of a model as part of a specific modeling language.

F

Function
A function is a projection of the behavior of the entire system (when seen as a black box) resulting in a relation between inputs and outputs with regard to a specific usage purpose.

Functional viewpoint
The functional viewpoint(↑) is a structured description of the functions(↑) that are to be realized along with their interfaces, interactions, and dependencies. The functional viewpoint imposes a structure onto the functional requirements of the SUD(↑). This viewpoint provides different model types that can be used for organizing hierarchies of system functions and the behavior of the system functions at the interfaces as well as their state space.

L

Logical component
A logical component is the result of a logical decomposition of a system into the internal logical structure of the SUD. It encapsulates a specific behavior that contributes to the realization of one or more functions(↑) of the SUD(↑). A logical component has a well-defined interface. A logical component may be decomposed into further logical components.

Logical viewpoint
The logical viewpoint is a structured description of how to organize the realization of the functions(↑) by means of logical components(↑) that are connected with one another. This viewpoint provides different model types that can be used for documenting the logical component architecture of a system, the behavior of the logical components at the interfaces, and their state space.

M

Mapping (between views(↑)/between abstraction layers(↑))
A mapping between views is a relationship between two models representing the views. A mapping can exist between models of different viewpoints(↑) or between models of the same view, but on adjacent abstraction layers(↑).

Model
A model is an abstract representation of an existing reality or a reality to be created. Every model is created for a specific purpose of use.

R

Refinement
Refinement refers to the process of detailing an analysis or design element while preserving its semantics.

Requirement
A requirement is:
1. A need perceived by a stakeholder
2. A capability or property that a system shall have
3. A documented representation of a need, capability, or property

[IREB 2011]

Requirements viewpoint
The requirements viewpoint is a structured description of how to document/specify the requirements of a system. This viewpoint provides different model types that can be used for documenting the system context(↑), system goals, system scenarios, and solution-oriented requirements.

S

(Operational) System context
The context of the SUD(↑) is the part of its environment that has an operational relationship to it during the execution of the system.

System under development (SUD)
The system under development is the subject that is being developed. Within the scope of the SPES engineering approach, the SUD refers to a software system.

T

Technical component
A technical component is a means for describing the technical structure of the SUD. It characterizes a technical resource that is available in the system and implements one or more logical components(↑) fully or in part.

Technical viewpoint
The technical viewpoint is a structured description of how to organize the realization of logical components(↑) by means of technical components(↑). The technical components may be related to each other. This viewpoint comprises different model types that can be used for documenting the hardware, tasks, and schedulers as well as the communication.

V

Validation
Validation refers to the activity of checking whether the requirements capture the stakeholder's needs and fulfill defined quality criteria. The goal is to assess whether a system that satisfies its defined requirements would fulfill its intended purpose. Hence, validation aims at answering the question: "Am I building the correct system?" [Boehm 1984]

Verification
Verification refers to the activity of checking whether a development artifact (e.g., the finalized SUD) satisfies the specified requirements. Hence, verification aims at answering the question: "Am I building the system correctly?" [Boehm 1984]

View
A view is a representation of a whole SUD from the perspective of a related set of concerns (based on [IEEE1471]).

Viewpoint
A viewpoint is a specification of the conventions for constructing and using a view(↑). Viewpoints comprise patterns or templates from which to develop individual views(↑) by establishing the purpose and audience for a view(↑) and the techniques for its creation and analysis (based on [IEEE1471]).

References

[IREB 2011] M. Glinz: A Glossary of Requirements Engineering Terminology. Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 1.1, May 2011.

[Boehm 1984] B. Boehm: Software Engineering Economics. Prentice Hall, New Jersey, 1984.

[IEEE1471] The Institute of Electrical and Electronics Engineers, Inc.: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std. 1471-2000. New York, 2000.

B – Author Index

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9,

A

Achatz, Dr. Reinhold

ThyssenKrupp AG
Corporate Center Technology,
Innovation & Quality
ThyssenKrupp Allee 1

45145 Essen, Germany

formerly:

Siemens AG
Corporate Technology
Otto-Hahn-Ring 6

81739 Munich, Germany iii

B

Beetz, Klaus

Siemens AG
Corporate Technology
Otto-Hahn-Ring 6

81739 Munich, Germany 3

Bender, Ottmar

CASSIDIAN
Woerthstrasse 85

89077 Ulm, Germany 15, 177

Böhm, Dr. Wolfgang

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3

85748 Garching, Germany 3

Broy, Prof. Dr. Dr. h.c. Manfred

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3

85748 Garching, Germany iii, 31, 251

D

Damm, Prof. Dr. Werner

Oldenburg Institute for Information
Technology (OFFIS)
Escherweg 2

26121 Oldenburg, Germany 31

Daun, Marian

paluno – The Ruhr Institute for Software
Technology
University of Duisburg-Essen
Gerlingstr. 16

45127 Essen, Germany 51, 119

Dieudonné, Laurent

Liebherr-Aerospace Lindenberg GmbH
Pfänderstraße 50-52

88161 Lindenberg, Germany 177

E

Eder, Sebastian

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3

85748 Garching, Germany 69, 85

F

Fasse, Dr. Friedrich-W.

RWE Consulting GmbH
Lysegang 11

45139 Essen, Germany 197

Fay, Prof. Dr. Alexander

Automation Technology Institute
Helmut Schmidt University Hamburg
Holstenhofweg 85

22043 Hamburg, Germany 137

Feilkas, Dr. Martin

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3

85748 Garching, Germany 69, 85

Fockel, Markus

Fraunhofer Institute for Production
Technology (IPT)
Zukunftsmeile 1

33102 Paderborn, Germany 157

G

Girod, Dr. Maurice

Airbus Operations GmbH
Im Kreetslag 10
21129 Hamburg, Germany 177

Glomb, Christian

Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany 197

Grünbauer, Johannes

SWM Services GmbH
Emmy-Noether-Straße 2
80287 Munich, Germany 197

H

Heidl, Peter

Robert Bosch GmbH
Corporate Research
P.O. Box 30 02 40
70442 Stuttgart, Germany 157, 243

Heinze, Hendrik

Berlin Heart GmbH
Wiesenweg 10
12247 Berlin, Germany 215

Henkler, Dr. Stefan

Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany 31, 95

Heuer, André

paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
Gerlingstr. 16
45127 Essen, Germany 197

Hilbrich, Robert

Fraunhofer Institute for Computer Architecture and Software Technology (FIRST)
Kekuléstr. 7
12489 Berlin, Germany 119

Hiller, Martin

CASSIDIAN
Woerthstrasse 85
89077 Ulm, Germany 15, 177

Höfflinger, Jens

Robert Bosch GmbH
Corporate Research
P.O. Box 30 02 40
70442 Stuttgart, Germany 157, 243

Höfig, Kai

Department of Computer Science
University of Kaiserslautern
Gottlieb-Daimler-Straße
67663 Kaiserslautern, Germany 107

Holtmann, Jörg

Department of Computer Science
University of Paderborn
Zukunftsmeile 1
33102 Paderborn, Germany 157

Hönninger, Harald

Robert Bosch GmbH
Corporate Research
P.O. Box 30 02 40
70442 Stuttgart, Germany iii, 157, 243

Horn, Dr. Wilfried

Hella KGaA
Beckumer Str. 130
59552 Lippstadt, Germany 157

J

Jäger, Tobias

Automation Technology Institute
Helmut Schmidt University Hamburg
Holstenhofweg 85
22043 Hamburg, Germany 137

Jedlitschka, Dr. Andreas

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany 131, 231

Jung, Jessica

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany 231

K

Kallow, Dr. Khalid

TeCNet Systeme & Service GmbH
Rudower Chaussee 29
12489 Berlin, Germany 215

van Kampenhout, J. Reinier

Fraunhofer Institute for Computer Architecture and Software Technology (FIRST)
Kekuléstr. 7
12489 Berlin, Germany 119

Klaus, Martin

SWM Services GmbH
Emmy-Noether Straße 2
80287 Munich, Germany 197

Kuntschke, Dr. Richard

Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
83719 Munich, Germany 197

L

Lackner, Harmut

Fraunhofer Institute for Computer Architecture and Software Technology (FIRST)
Kekuléstr. 7
12489 Berlin, Germany 215

Lampasona, Dr. Constanza

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany 231

Laskowski, Prof. Dr. Michael

RWE Deutschland AG
Kruppstraße 5
45128 Essen, Germany 197

Liggesmeyer, Prof. Dr. Peter

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

and

Department of Computer Science
University of Kaiserslautern
Gottlieb-Daimler-Straße
67663 Kaiserslautern, Germany 107

Löwen, Dr. Ulrich

Siemens AG
Corporate Technology
San-Carlos-Str. 7
91058 Erlangen, Germany 131, 137

M

Meyer, Dr. Jan

Hella KGaA
Beckumer Str. 130
59552 Lippstadt, Germany 157

Meyer, Dr. Matthias

Fraunhofer Institute for Production Technology (IPT)
Zukunftsmeile 1
33102 Paderborn, Germany 157

Mund, Jakob

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3
85748 Garching, Germany 85

P

Pohl, Prof. Dr. Klaus

paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
Gerlingstr. 16
45127 Essen, Germany iii , 31

R

Ratiu, Dr. Daniel

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3
85748 Garching, Germany 69

Reinkemeier, Philipp

Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany 95

S

Sadeghipour, Dr. Sadegh

ITPower Solutions GmbH
Kolonnenstraße 26
10829 Berlin, Germany 215

Schäuffele, Jörg

Vector Informatik GmbH
Ingersheimer Straße 24
70499 Stuttgart, Germany 157

Schlingloff, Prof. Dr. Holger

Fraunhofer Institute for Computer Architecture and Software Technology (FIRST)
Kekuléstr. 7
12489 Berlin, Germany 215

Schuller, Peter

MicroSys Electronics GmbH
Muehlweg 1
82054 Sauerlach, Germany 137

Sojer, Dominik

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3
85748 Garching, Germany 119

Stierand, Dr. Ingo

Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany 95

Strobel, Carsten

EADS Innovation Works
Willy-Messerschmitt-Straße 1
85521 Ottobrunn, Germany 177

T

Tahirbegovic, Salko

T+I Technologie- und InnovationsConsult GmbH
Schlaatzweg 1
14773 Potsdam, Germany 215

Tenbergen, Bastian

paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
Gerlingstr. 16
45127 Essen, Germany 15, 51, 243

Trapp, Dr. Mario

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany 107

V

Vogelsang, Andreas

Department of Informatics
Technische Universität München (TUM)
Boltzmannstr. 3
85748 Garching, Germany 31, 69, 85

W

Wagner, Dr. Thomas

Siemens AG
San-Carlos-Str. 7
91058 Erlangen, Germany 137

Waßmuth, Martin

EADS Innovation Works
Willy-Messerschmitt-Straße 1
85521 Ottobrunn, Germany 177

Weber, Raphael

Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany 95

Wehrstedt, Dr. Jan Christoph

Siemens AG
Otto-Hahn-Ring 6
81739 Munich, Germany 137

Weyer, Dr. Thorsten

paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen
Gerlingstr. 16
45127 Essen, Germany 15, 31,51,119, 197

Wiesbrock, Dr. Hans-Werner

IT Power Consultants
Gustav-Meyer-Allee 25
13355 Berlin, Germany 215

Z

Zimmer, Bastian

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany 107

C – Project Structure

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9, © Springer-Verlag Berlin Heidelberg 2012

Co-operation

The innovation alliance focused on a strong interaction between science and practice in order to make good progress in the engineering challenges, to verify the approaches, and to transfer the results to engineering methods that are viable in practice. This target was supported by the following:

- Significant application projects from five application areas showing high dynamics and with high economic relevance
- Installation of a central project that interacted with the application projects and provided the necessary methods and tools
- Installation validation work packages in which the results were evaluated using scientific taxonomy

The structure between the different locations in Germany allowed for the inclusion of key resources into the innovation alliance. The close co-operation between the industrial partners and the universities/Fraunhofer institutes has ensured that the results could be evaluated with respect to practical and market-feasible solutions.

Project structure

As stated above, SPES 2020 was structured as a central project and five application projects. The application projects correspond to the SPES 2020 domains automation, automotive, avionics, energy, and healthcare.

The central project itself was divided into work packages one to six. Packages two to five deal with the following topics:

- Model-based requirements engineering
- Model-based architecture design
- Proof of safety, certification, and quality assurance of nonfunctional requirements
- Modeling of parallel real-time processes and verification of the real-time behavior

The foundation was laid in work package one, which developed a methodology for a comprehensive and integrated model-based development. In work package six, the focus was on the empirical evaluation of the methods.

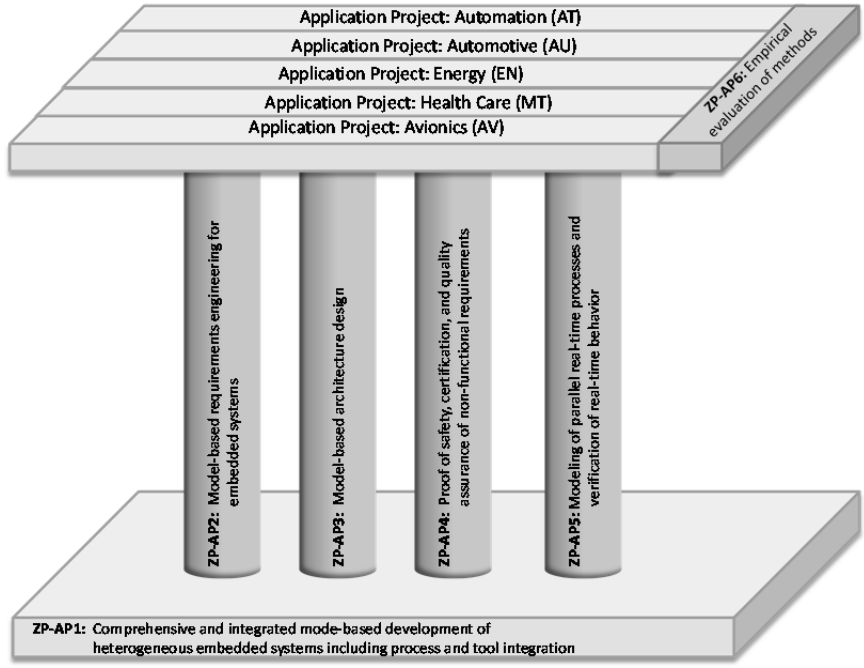


Fig. C *SPES 2020 project structure*

The interplay between the work packages of the central project and the application projects was assured by a strong orientation on case studies from the application domains.

D – Members of the Innovation Alliance

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9, © Springer-Verlag Berlin Heidelberg 2012

Airbus Deutschland GmbH

Airbus Deutschland GmbH is one of the world's largest manufacturers for civil aircraft seating more than 100 passengers and Europe's largest industrial undertaking. With some $11 billion in annual revenues, it has won approximately 50% of all orders for jet-powered transport airplanes in recent years. Airbus Deutschland GmbH, the civil aircraft business affiliate of Airbus S.A.S. – Toulouse, is one of the major aerospace companies in Europe. The company is engaged in the development and manufacture of advanced high-performance commercial transport aircraft and is recognized for its technological expertise in nearly all fields of aeronautical engineering. The workforce of Airbus Deutschland currently amounts to approx. 18,500 employees. Research and technology development capabilities of Airbus Deutschland GmbH cover all aspects of aircraft design and optimization, airframe and systems development, and engine integration. For systems integration, IMA (Integrated Modular Avionics), Airbus D has experience in integration of cabin systems through the A380 and will apply and improve this technology in future.

For the SPES 2020 project, Airbus provided a showcase development scenario (architectural framework) from the aircraft systems area as a deployment example for the SPES 2020 development platform. Modeling languages for system requirements were analyzed and checked for integrity characteristics. The requirements for a development platform were formulated from an avionics point of view and the resulting process analyzed. The avionics example scenario was implemented on the development platform and evaluated with respect to processes and tools available at Airbus. Results were validated and potential for improvement delivered by the platform analyzed for future industrial deployment.

Berlin Heart GmbH

Berlin Heart GmbH is the only company worldwide developing, manufacturing, and selling internal and external heart support systems for patients of all ages and all body heights.

These systems support patients with major cardiac insufficiency. As life-sustaining systems, these are medical products in the highest risk level (level III). All components for control and monitoring based on embedded control systems are implemented as safety-critical components. Therefore, especially high standards with regard to requirements management, development, implementation, testing, production, and certification apply.

The Berlin Heart products INCOR® und EXCOR® are market leaders in their respective segments in Germany and in Europe. Employing over 180 people, and doubling revenue to €22 million within three years in 2007, the company has an outstanding performance.

Cassidian (EADS-Deutschland GmbH Defence Electronics)

Cassidian (http://www.cassidian.com) is a global leader in aerospace, defense, and related services. In 2007, EADS generated revenues of €39.1 billion and employed a workforce of approximately 116,000. The group includes the aircraft manufacturer Airbus, the world's largest helicopter supplier Eurocopter, and EADS Astrium, the European leader in space programs from Ariane to Galileo. Its Defence & Security Division is a provider of comprehensive systems solutions and makes EADS the major partner in the Eurofighter consortium, as well as a stakeholder in the missile systems provider MBDA. EADS also develops the A400M through its Military Transport Aircraft Division.

The Cassidian group encompasses inter alia the business unit Defence Electronics (EADS-DE). The high technological competence and experience is founded on a long tradition of famous pioneering companies, mainly in Germany and France. As the sensors, avionics, and electronic warfare house of EADS, Defence Electronics unites advanced sensor and electronic technologies for all types of platforms—manned and unmanned aircraft, helicopters, satellites, vehicles, ships—and provides them with components and subsystems based on the latest radar, electronic, and software technologies. Recent activities covered special equipment on board the A380, demanding electronic systems with different safety levels. Current developments include several computer and communication systems on board new aircraft subject to highest civil certification levels following RTCA/DO 178B level A to C and the corresponding DO 254.

Furthermore, Defence Electronics develops and manufactures mission avionics and self-protection systems, which are mostly established in international co-operations and multinational projects.

Cassidian brings the competency of developing complex, performance- and safety-critical embedded systems to the project. This competency results from the product portfolio that covers embedded systems aircraft, shipping, and ground systems. In SPES 2020, Cassidian concentrated on safety, certification, and quality, as well as real-time and multicore architectures, bringing in its expertise from actual and completed projects. Leading the avionics application project, Cassidian cooperated closely with its partners Airbus, EADS-IW, and Liebherr Aerospace, supporting the academic partners in the avionics subproject.

In addition, researchers from Cassidian were members of the project architecture team.

EADS-Deutschland GmbH Innovation Works (EADS-IW)

EADS Innovation Works (IW) is the corporate research facility of EADS with operations in France, Germany, Spain, UK, Singapore, and Russia. Its overall workforce comprises more than 700 researchers. EADS-IW covers all the skills and technology fields that are of critical importance to EADS. It is an operational and strategic entity for EADS business units for value-creating products and services through innovative technologies. It feeds the innovation pipeline from the emergence of new technologies to their maturity and transfer

into products. EADS-IW, in its endeavor to maximize the innovation potential of EADS, actively operates a worldwide network with world-class universities, schools, and research institutes. In its legal structure, EADS-IW is part of the national entities of EADS.

The German part of EADS Innovation Works in Ottobrunn (near Munich) and Hamburg employs a permanent staff of 220 people, 70% of which are senior scientists. Legally, it is an organizational unit within EADS-Deutschland GmbH, the German subsidiary of EADS N.V.

In SPES 2020, EADS-IW focused on transforming the scientific research results into practical innovations. Working closely together with its partners (e.g., Cassidian and Airbus), EADS-IW defined case studies in SPES 2020 and identified problem areas that partners in the central project worked on . EADS-IW was also engaged in the central project, preparing concrete deliverables, together with its partners, covering topics such as modeling framework, requirements modeling and validation, design verification and formal analysis, as well as technology platform.

EADS-IW had the role of coordinating the industry partners in ZP-AP 3, offering a strong background in systems and software engineering, engineering frameworks, and from other sponsored projects such as VIVACE (Aeronautic) and SPEEDS (Information Technology).

Embedded4You e.V.

Embedded4You offers integrated solutions for embedded systems in industry automation that set the standards for the future by leveraging the individual competencies of its members. The member companies Afra, aicas, Coming, Elma Trenew, Euro Systems, FH München, Fortiss, FTI Group, ISyst, Kölsch & Altmann, Microsys, N.A.T., IMACS, Protos Software GmbH, RST Industrie Automation, sepp.med, Tieto embedded Systems, and XiSys bundle their individual technologies, products, services, and competencies into a comprehensive total package for automation challenges. Through dedicated project leadership, customers get their complete solution from one source.

Embedded4You system solutions, starting with design, via product and go to market, through to life cycle management, enhance the competitiveness of customers with their openness and flexibility. The following members were active in the SPES project:

Elma Electronic GmbH is a global manufacturer of electronic packaging products for the embedded systems market — from components, storage boards, back-planes, and chassis platforms to fully integrated subsystems. The company has a broad base of proven standard products that can be tailored to individual applications, from initial concept to volume production. Elma's reliable solutions, flexibility, and design expertise make Elma a reliable partner for leading electronics companies in the world.

The *FTI Group* success story began with FTI (Flight Test Instrumentation) — the development of test systems. The company is now one of the largest engineering services providers and system developers in the aerospace region Berlin-Brandenburg, specializing

in system solutions for aviation. Development, design, and consulting in the high-tech industry and energy sector complete the portfolio.

IMACS GmbH develops and produces instrumentation, control, and automation systems for various industries. Whether single components or complete embedded systems, the solutions are always individual and flexible. In addition, IMACS offers radCASE, a model-based software development system that allows for the comprehensive development of technical software, including code generation.

MicroSys Electronics GmbH, located in Sauerlach close to Munich, designs and develops embedded system solutions, for example, VMEbus, CompactPCI, and other common bus infrastructures. From the very beginning in 1975, customized solutions offering longevity have been a strong part of MicroSys business as well. Successfully deployed products span from computer-on-modules up to fully integrated systems. The miriac™ Modules utilize 32-bit processors such as Freescale Power Architecture, QorIQ-CPUs, Intel MultiCore CPUs, DaVinci ™ Video Processors, FPGAs, and DSP-Designs. With their low power consumption and the compact dimensions of a credit card, the miriac™ CPU modules fit into any application in automotive, industrial automation, medical, railways and transportation, construction, and defense market segments.

Operating systems such as VxWorks, Microware OS-9, Micrium µC/OS, QNX, and Linux or WinCE are supported. Furthermore, MicroSys acts as a sales and support partner in Europe for RadiSys Microware ® OS-9 Real Time Operating System.

Operating system integration and adaptations of communication infrastructures such as CAN, EtherCAT, ProfiNET for industrial, defense, avionics, and medical solutions are an integral part of the business as well.

N.A.T. (Gesellschaft für Netzwerk und Automatisierungs-technologie mbH) is the expert in manufacturing high-performance connectivity products for data and telecommunication solutions. The product portfolio is dedicated to the embedded market, covering requirements from local area networks (LAN) up to wide area networks (WAN). The products include standard interface modules for local and wide area networks based on common hardware standards such as AMC, MicroTCA, VME, compact PCI, PMC, PCI, and others. N.A.T. embedded platforms are complemented by own, sophisticated protocol stack solutions such as ISDN, SS7, ATM, or TCP/IP adapted to common real-time operating systems to build an optimal solution.

RST Automation GmbH builds embedded systems and applications with a focus in industry automation for reliable real-time requirements. The products and solutions are open for easy integration of existing and new technologies. Based on the model-oriented approach of the middleware "Gamma," which can be seen as a type of "software plug," it is possible to describe and integrate virtually any hardware and software configuration. The middleware is used to integrate hardware and software into one common platform. Within Embedded4You, the middleware is the main strategy for integrating custom-specific efforts into one homogenous platform.

XiSys Software GmbH manufactures the tool XiBase9, a well-structured and portable graphic platform that has been specifically developed for the embedded and real-time

environment. Highlights of the software are its efficient resource usage, multilanguage support (Unicode, language switch at runtime), and the extensive protocol and debug mechanisms. The software is operating system-independent (Windows 2000, XP, Vista, Linux, OS-9, x86, PowerPC, SH, ARM, XSCALE, 68k) and can be connected to the Gamma middleware without programming.

Taking its role as a supplier of specialized solutions, Embedded4You's goals in SPES were professionalization of the interdomain production process by focusing on basic, interapplication approaches to leverage the potential of embedded systems and to master their complexity. The work focused on the development of an open platform that integrates and standardizes various key competencies in hardware and software products.

Hella KGaA Hueck & Co.

The automotive supplier Hella KGaA Hueck & Co., Lippstadt, develops and manufactures lighting technology and electronic components and systems for the automotive industry. In addition, joint venture companies produce complete car modules such as air conditioning systems and on-board electrical systems. Hella owns one of the worldwide largest sales organizations for car parts and accessories, with own sales businesses and partner organizations in more than 100 countries. The Hella group has revenues of €3.7 billion.

Hella is among the top 50 international automotive suppliers and among the top 100 largest companies in Germany. Worldwide, more than 25,000 people work in 70 production sites, production subsidiaries, and joint ventures in 18 countries. More than 3000 engineers and technicians work in research and development.

Hella's customers include all leading vehicle and system manufacturers, as well as car parts businesses. In the electronics business area, but also in the growing electronic share of lighting technology products, Hella has built up a broad competency spectrum and deep experience in the development of complex mechatronic systems for the automotive industry. From the latest methods in model-based software development—in SPiCE-compliant development processes using AUTOSAR standard components—numerous crosscutting development approaches for embedded systems in the automotive industry have been defined together with customers and cooperation partners. Hella's main support for the project was in defining, detailing, implementing, evaluating, and optimizing concepts in the automotive work package, and via the definition of area-specific requirements, the company also supported the work of the central project.

IT Power Consultants

IT Power Consultants, based in Berlin, was founded in 2000 and recently changed its name to ITPower Solutions GmbH. Longtime experience of the founders in the area of embedded systems established the company's focus on development processes for embedded software.

Motivated by experiences from customer projects, the company constantly invests in the improvement of the development and testing processes, especially in the area of quality assurance of embedded systems.

In SPES 2020, IT Power Consultants contributed in the areas of requirements engineering and verification of real-time behavior in the medical systems work package, most notably covering the following topics:

- Transition from architectural model to functional model of an embedded system
- Proof of equivalence of software artifacts by back-to-back testing
- Identification of test cases on the basis of nonfunctional requirements
- Hardware-in-the-loop environments for testing real-time properties of embedded systems

Liebherr-Aerospace Lindenberg GmbH

The company founded in 1949 by Hans Liebherr is today a group of companies with more than 32,000 employees in more than 120 enterprises worldwide. Turnover in 2010 was €7.5 billion. All over the world, the name Liebherr stands for a technically demanding and customer-oriented product and service offering.

Liebherr-Aerospace, a worldwide respected supplier of the aviation industry, develops and produces complete hydraulic, mechanical, and electronic systems for cruise control, air conditioning, and landing gear for large and regional aircraft, helicopters, and military aircraft for the global market. Liebherr-Aerospace customers include Airbus, Bombardier, Embraer, Sukhoi, Eurocopter, and others.

In past and current research projects, Liebherr-Aerospace Lindenberg GmbH has contributed to all product areas that laid the foundation for new equipment systems. In addition to the national Luftfahrtforschungs-Programm (LuFo I to IV), on a European level, the 5th, 6th, and 7th framework programs should be mentioned. The generic results from these projects have been transferred to national value creation potentials.

Liebherr-Aerospace brought competencies in the areas of flight control systems and landing gear from pilot interface to hydraulic or electronic actuators into the SPES project. Special focus was on quality of requirements of complex systems — these are key for competitiveness. Therefore, Liebherr-Aerospace contributed to the realization of the SPES 2020 follow-through modeling methodology, especially for the avionics domain, and to the investigation for validation of requirements in early development phases. As a major engagement, Liebherr-Aerospace has proposed an original and efficient approach to optimizing the deployment of functions on computer networks. Liebherr-Aerospace has investigated the improvement of the automation of the verification activities in accordance with the restrictive avionics development standards. In addition, the company produced proposals for the efficient use of multicore platforms for real-time and safety-critical applications.

Robert Bosch GmbH

Bosch is one of the largest industrial enterprises in Germany, creating revenues of €46.3 billion (2007). In the business areas automotive engineering (61% revenue), industrial engineering (13% revenue), and consumer goods and building services engineering (26% revenues), Bosch employs approximately 271,000 employees.

The corporate law structure of Robert Bosch GmbH ensures the corporate independence of the Bosch Group and enables the company's long-term planning. The nonprofit Robert Bosch Foundation GmbH holds 92% of the capital shares.

Bosch sees professional education as part of its social responsibility. Year after year, more than 6000 young people (of those, approximately 4400 in Germany) receive an offer for a high-quality training program. The Bosch Group invests approximately 7.7% of its revenue in research and development, with 29,000 employees. In 2007, 3280 patents were filed worldwide.

At Bosch, protection of the environment is in line with corporate policy. Protection of the environment was formulated as a company target as early as 1973, and has the same significant value as product quality and profit. It is Bosch's vision to improve quality of life with innovative and useful solutions. "Reliability, credibility and legality are the main factors of the economic success of the Bosch-Group," said Hermann Scholl, Chairman of the Board.

The central Corporate Research (CR) team works across business areas and therefore provides a competency network that warrants the development of innovative system concepts as well as the introduction of new technologies.

Focusing necessarily on the competencies, CR/AE2 is a group responsible for the development of software-intensive embedded systems. It develops, integrates, and pilots lead applications and successfully transfers technologies, methods, processes, and tools necessary for the development of software-intensive systems within Bosch, thus warranting Bosch's leading position in the market. The CR/AE2 group brought system design know-how in using cutting edge methodologies for modeling and evaluation, which has been developed in numerous projects, into the SPES project. Therefore, the main focus areas were on "seamless model-based development of heterogeneous embedded systems including process- and tool-integration" (ZP AP1), "structured requirements engineering" (ZP AP 2), "safety and certification" (ZP AP 4), and "empirical evaluation" (ZP AP 6). Here, Bosch provided know-how and the results have already been applied to concrete applications.

In addition, researchers from Bosch were members of the project architecture team.

RWE Energy AG

RWE Energy, based in Dortmund, bundles the integrated sales of electrical power, gas, and water, as well as the network business for 12 regions in Germany and Continental Europe.

RWE Energy employs 28,323 people in Germany, Austria, Hungary, Slovakia, Poland, and the Netherlands.

As a leading player, RWE Energy aligns itself with the requirements of 23.1 million customers. In Germany and wide parts of Central Europe, prosumers of RWE Energy get their energy and water solutions from one shop, making RWE Energy a number one address for all questions concerning energy and water supply, with €28.2 billion annual revenues.

Annual sales of 168.3 billion kilowatt hours of electrical energy makes RWE Energy the number two in Germany and number three in Europe. In water supply, RWE Energy is the number one in Germany, selling 107 million cbm per year. Gas sales are 258 billion kilowatt hours annually.

With two focal points, sales and network, RWE Energy brought deep insight from energy supply areas, as well as practical and theoretical knowledge of planning and running energy networks, into the SPES project. In addition, RWE Energy was able to leverage know-how from research projects and internal projects already completed and covering smart grids and virtual networks.

Siemens AG

Siemens is a global powerhouse in electronics and electrical engineering, operating in the fields of industry, energy, and healthcare, as well as providing infrastructure solutions, primarily for cities and metropolitan areas. For over 160 years, Siemens has stood for technological excellence, innovation, quality, reliability, and internationality. The company is the world's largest provider of environmental technologies. Around 40 percent of its total revenue stems from green products and solutions. In fiscal year 2011, revenue from continuing operations totaled €73.5 billion, and net income from continuing operations €7.0 billion. At the end of September 2011, Siemens had around 360,000 employees worldwide on the basis of continuing operations. The company employs some 27,800 researchers and developers worldwide who work on innovations that secure existing business and open up new markets. In fiscal year 2011, Siemens invested €3,925 million in research and development. In the same period, the employees submitted around 8,600 invention reports — around 40 per workday. The role of Siemens AG in the project was manifold:

- Firstly, Siemens adopted the role of a developer and manufacturer of automation equipment including corresponding software. Siemens provides a wide spectrum of automation components, as well as control and management systems for the various industry segments (process industries, discrete industries, service industries), holding a worldwide number 1 position in many segments.
- Secondly, Siemens was active in the role of a system integrator, equipment manufacturer, and turnkey contractor respectively. Siemens is an engineering company offering customer-specific facilities, plants, and automation solutions for selected

industry segments (e.g., automotive, building automation, electric power generation, transmission, distribution, metals and mining, chemical, transportation and logistics).

- The third role of Siemens AG was that of a healthcare solution provider with key competencies and innovation strength in diagnostic and therapeutic technologies, as well as an integrated supplier of data processing solutions for the whole clinical chain.

Together with a broad know-how in product life cycle management (PLM) software, Siemens covers all necessary competencies for industrial product and solution development using embedded systems — from product development to production design and engineering to generation of automation software.

In addition, researchers from Siemens were members of the project architecture team.

SWM Services GmbH

SWM Services GmbH is a 100% subsidiary of Stadtwerke München GmbH (SWM), acting as internal service provider for services in the areas energy data, trade fair services, network and facility services, as well as information and process technology.

Stadtwerke München, Munich's municipal utilities company, is one of the largest energy and infrastructure companies in Germany. Over one million private households, SMEs, and business clients benefit from the services provided by SWM on a daily basis. For decades, SWM has provided energy (electricity, natural gas, district heating) for the Bavarian capital in a safe and environmentally benign way. Among other things, the SWM development push for renewable energy and the push for eco-friendly district heating are an example to other districts. Furthermore, SWM supplies the megacity with fresh drinking water from the Bavarian Voralpenland—one of the best in Europe—and with 18 indoor and outdoor swimming pools, SWM operates one of the most modern bathing environments in Germany. The MVG transport subsidiary is responsible for the subway, bus, and tram systems, and is therefore a significant pillar in Munich's public transport network. SWM employs around 7500 staff and in the 2010 fiscal year, turnover reached around €3.8 billion.

As a public utility company, SWM Services GmbH has broad experience in all supply areas, especially in the energy area. In addition, SWM has in-depth knowledge in model-based system and software development, and brings first project experience in the area of embedded systems for smart metering and virtual power plants into the SPES project.

Being involved in the energy application area, SWM leveraged this experience and actively supported the definition and validation of the application-specific requirements. Another focus area was the implementation and evaluation of an integrated simulation environment for smart grid development. The work of the central project was supported by the execution of practicability tasks.

TeCNeT GmbH

TeCNeT, organization for TeleCooperate NeTwork Systems & Service mbH, was founded in 1994 as a service provider and developer of innovative products. Its main competencies are in the areas of information and telecommunication technology, as well as in medical systems and control engineering.

TeCNeT offers a full range of services from technical consulting, to support for development and manufacturing, to full-service offerings from the idea to ready-to-use implementation of the solution. Its main strengths are flexibility, speed, and cost-efficiency, which can be measured by the satisfaction of TeCNeT's customers.

Customers include: CeWe Color AG & Co. OHG, the Fraunhofer Institut für Fabrikautomatisierung und Fabrikbetrieb, Gesellschaft für angewandte Informatik (GfaI), Berlin Heart GmbH, as well as other smaller regional businesses.

TeCNeT's contribution to SPES 2020 comprised of leveraging the existing know-how for the development of embedded systems for supervision of patients and the medical devices necessary for maintaining their health.

TÜV SÜD AG

TÜV SÜD group is a future-oriented and successful service business. 13000 employees support more than 5 million customers (individuals, companies, and institutions) worldwide.

The subsidiary TÜV SÜD Automotive GmbH is a modern service company providing a complete portfolio, for example, in the area of safety electronics for automotive and electronic industry sectors. With its competence center Electronic Safety, TÜV SÜD Automotive GmbH is active as an intersectoral support and test center for safety electronics with a main focus on functional safety and software.

In this role, TÜV SÜD Automotive GmbH consults, tests, and certifies manufacturers of safety-related embedded systems. Experience from test and certification of safety-related embedded systems (using model based technologies) was brought in into the SPES project. TÜV SÜD Automotive GmbH has made sure that the technologies developed will be testable and certifiable in accordance with international (safety) standards.

Vector Informatik GmbH

Vector supports manufacturers and suppliers of the automotive industry and of associated businesses with a professional and open platform composed of tools, software components, and support services for the development of embedded systems. The know-how is offered in terms of products, as well as a holistic support-offering including systems and software engineering. Workshops and seminars complete Vector's portfolio.

Vector Informatik GmbH is part of the Vector group that includes other companies in Germany (Vector Consulting Services GmbH, aquintos GmbH), France (Vector France S.A.S), Sweden (VecScan AB), United Kingdom (Vector GB Limited), China (Vector Automotive Technology Co., Ltd.), USA (Vector CANtech, Inc.), Japan (Vector Japan Co., Ltd.), South Korea (Vector Korea IT Inc.), and India (Vector Informatik India Pvt. Ltd.). The Vector group employs more than 1000 people and achieved revenues of €195 million in 2011. All sites of the Vector group have been certified according to ISO 9001:2000.

In SPES 2020, Vector worked in the automotive work package and contributed practical expertise in developing embedded systems.

In the product line "Process Tools," Vector is developing the PREEvision tool. PREEvision supports the model-based development of electric/electronic systems from the early architecture design to production maturity of embedded systems.

In the automotive industry, embedded systems are mainly being developed in terms of a platform or product line concept. Vector therefore contributed to the work package "Variability management for model-based development" and provided experience from the practical deployment of PREEvision in the automotive industry

Fraunhofer FIRST

The Fraunhofer Institute for Computer Architecture and Software Technology (FIRST) was founded in 1983 as an institute of the Society for Mathematics and Data Processing (GMD), and has been part of the Fraunhofer-Gesellschaft since July 2001. Today, some 140 employees work in the three departments Modeling, Systems Architecture, and Quality Assurance.

Researchers at Fraunhofer FIRST combine long-standing know-how of hardware architectures and software methods with extensive skills in quality assurance in order to advance safety, efficiency, and usability of embedded systems. Main goals are the development of premium, easy-to-use, and intelligent technologies that adapt to the user's needs and support them optimally. To achieve these goals, Fraunhofer FIRST develops innovative methods and technologies and advises companies during the entire development chain: from modeling to architecture and quality control to the completed product.

The research group is concerned with real-time-capable, reliable, and secure integration of multicore processors into embedded systems. It aims at reducing the complexity of developing efficient embedded multicore systems, while at the same time exploiting performance potentials and maintaining the reliability of the overall system. Multicore systems should remain real-time-capable and energy-efficient even under optimal workloads.

The research group Embedded Systems advises clients on the selection of methods and tools, architectural designs, prototypical implementation of components and subsystems, and on the evaluation, test, and certification of the embedded multicore systems.

In the SPES context, the following key competencies of FIRST researchers were most relevant:

- Model-based test generation and execution
- Verification and static analysis of industrial customer software
- Fault tolerance concepts and reliability quantification
- Architecture of embedded control elements, FPGA, System-On-Chip
- Comprehensive tool know-how for the development of embedded systems.

With these competencies, FIRST led the work package ZP-AP5 “Real-Time and Safety” within the central project and made major contributions in AWP-MT.

Fraunhofer IESE

Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern is one of the worldwide leading research institutes in the area of software and systems development. A major portion of the products offered by its collaboration partners is defined by software. These products range from automotive and transportation systems, through automation and plant engineering, information systems, healthcare, and medical systems, to software systems for the public sector. The solutions allow flexible scaling. This makes the institute a competent technology partner for organizations of any size — from small companies to major corporations.

Under the leadership of Prof. Dieter Rombach and Prof. Peter Liggesmeyer, Fraunhofer IESE has spent the last fifteen years making major contributions to strengthening the emerging IT hub of Kaiserslautern. In the Fraunhofer Information and Communication Technology Group, it cooperates with other Fraunhofer institutes in developing trend-setting key technologies for the future.

Fraunhofer IESE is one of 60 institutes of the Fraunhofer-Gesellschaft. Together they have a major impact on shaping applied research in Europe and contribute to Germany’s competitiveness in international markets.

The work at Fraunhofer IESE focuses mainly on methods for the development of software-intensive embedded systems as well as the empirical evaluation of such methods.

Fraunhofer IESE has been engaged in the engineering-like development of embedded software and systems for more than a decade, and has proven itself as one of the leading research institutes worldwide. One of its internationally respected unique selling propositions is the empirical evaluation of research results in the area of software and systems engineering. This competency was brought to the project by Fraunhofer IESE taking the scientific lead of the work package ZP-AP6 “Empirical Methods Evaluation” within the central project.

A second focus of the institute was on the model-based development of safe and highly reliable embedded systems. Given this expertise, Fraunhofer IESE led the workpackage ZP-AP4 “Proof of Safety, Certification, and Quality Assurance of Nonfunctional

Requirements." In this role, the institute developed methods, techniques, and tools for proof of safety, certification, and quality assurance in SPES.

In addition, researchers from IESE were members of the project architecture team that made major contributions towards an integrated model-based methodology.

OFFIS e.V.

The "Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und –Systeme," OFFIS for short, was founded in 1991 and has a close cooperation agreement with the University of Oldenburg. OFFIS sees itself as an application-oriented research and development institute, and as "Center of Excellence" for selected topics in computer science and its application domains. OFFIS focuses its research and development work on IT systems in the application areas transportation, healthcare, and energy. Revenue is approximately €12 million.

In SPES 2020, the R&D division Transportation was involved. Its research focuses on methods, tools, and technologies for the development of reliable, cooperative, and supporting systems in the application area transportation. The division comprises several working departments, and offers a wide spectrum of competencies in the areas systems and software engineering, electrical engineering, and planning theory. Main research topics include methods, processes, and tools for the establishment of safety in transportation systems, as well as methods for analysis and design of E/E architectures. A special focus is on real-time aspects and component-based design.

OFFIS has participated in numerous national and European research projects, including OPRAIL, Verisoft, SafeAIR, SPEEDS, COMBEST, ArtistDesign, ESACS, ISAAC, and MISSA, and continues to do so today. Because of its competencies in the area of real-time, OFFIS is also a partner in AUTOSAR and, via SafeTRANS, a member of EICOSE, the ARTEMIS Innovation Cluster on transportation.

In SPES, OFFIS led the work package ZP-AP3 "Model-Based Architecture Development" within the central project and participated in the work packages ZP-AP1, ZP-AP4, and ZP-AP5 with a focus on the range of topics around "model-based architecture design." This included the development of an integrated, cross-domain approach that can be adapted to the specific requirements of the respective application domains. In addition, researchers from OFFIS were members of the project architecture team that made major contributions towards an integrated model-based methodology.

As a partner in the ARTEMIS project CESAR, OFFIS arranged for synergies related to the CESAR Reference Technology Platform.

University of Kaiserslautern

Software Engineering Research Group: Dependability

The research conducted by the research group Software Engineering: Dependability at the faculty of Computer Science at the Technical University Kaiserslautern focuses on methods for developing embedded software that meets high quality standards. Current goals concern object-oriented methods, especially with respect to applications in safety-critical, highly available real-time systems. In particular, the research considers the ongoing growth of software and its distributed architecture. Many projects are conducted in collaboration with industrial partners.

In the SPES 2020 context, the research group Software Engineering: Dependability worked on the question of how software engineering and safety engineering can be interweaved more closely, e.g., by automatically deriving safety models from the software design and—vice versa—the integration of measures into the software development process to increase safety.

Software Engineering Research Group: Processes and Measurements

The research group Software Engineering: Processes and Measurement at the faculty of Computer Science at the Technical University Kaiserslautern has its expertise in modeling and quantitative prediction of integrated development processes, as well as in empirical analysis of software development methods and tools.

Today, software development projects are characterized by overshooting time and budget limits significantly. The main reason for this is insufficient knowledge of the potentials and limits of particular development methods and tools in a concrete project environment. On the other hand, process modeling methods that would allow coordination of the activities of the individual members of the development team, as well as a progress control with regard to content, are missing.

The work of the research team in SPES focused on developing methods to model complex development processes and to instrument them for prediction and progress control, as well as on the empirical analysis of individual methods and tools.

Technische Universität München (TUM)

TUM is one of the leading universities in Germany. TUM’s top performances in research and education, interdisciplinary studies, and talent promotion stand out. Strong alliances with businesses and scientific institutions across the world play a part in this. TUM was one of the first "Universities of Excellence" of the nationwide Excellence Initiative and impressed this cooperative in 2006 with its concept of "TUM. The Entrepreneurial University.”

The Department of Informatics attaches high importance to a close link between scientific research and study. The systems and tools developed there are constantly being tested by students and research staff in a practical deployment. In SPES, the chairs Software & Systems Engineering (Prof. Broy) and Embedded Systems & Robotics (Prof. Knoll) were engaged.

Software & Systems Engineering chair

The research and teaching efforts of Prof. Broy's chair Software & Systems Engineering are centered around core topics of software and systems development. This includes foundations, methods, processes, models, description techniques, and tools.

The research focuses on development of critical embedded systems, mobility and context-awareness, and development methods for complex industrial-scale software systems. The methods are supported by a number of research tools. Theorem-proving techniques explore the foundational aspects of software engineering. The methods developed in the group have been validated in various industry cooperations in the telecommunications, avionics, automotive, banking, and business information systems domains.

The chair is involved in an extensive number of basic and applied research projects. Additionally, it offers targeted enterprises specific consulting services, and develops tool prototypes and demonstrators. The Software & System Engineering chair took the overall project lead for the entire SPES project. With regard to content, the research focus was on developing an integrated modeling theory in the context of the SPES central project. Several tools for the development of embedded systems have been provided. Therefore, besides the research of the theoretical modeling theories, these practical competencies were also brought into the project. In addition, researchers from TUM were members of the project architecture team that made major contributions towards an integrated model-based methodology.

Embedded Systems and Robotics chair

The primary mission of the Embedded Systems and Robotics chair is research and education of machines for perception, cognition, action, and control. The chair is organized into four research areas:

Human Robot Interaction and Service Robotics, including work on the integration of speech, language, vision, and action; programming service robots; development of new application scenarios for sensor-based service robots; robot systems for education;

Medical Robotics, covering all aspects of manipulator and instrument control for complex surgical procedures, e.g., visualization of all types of patient data, haptic feedback for delicate handling, skill transfer, shared control, multimanipulator cooperation;

Cognitive Robotics, encompassing a comprehensive area of topics ranging from sensor models by the way of individual sensor processing entities (e.g., for high-speed face tracking) to high-level cognitive skills for navigation, adaptation, learning;

Cyber-Physical/Embedded Systems are investigated with special emphasis on fault tolerance and high availability; special topics are the design of very small redundant systems and the associated software development models and tool chains.

In SPES, the contribution was to provide leading edge technology for developing solutions in the automation area. The focus was on the development of domain-specific tools for the generation of nonfunctional properties (QoS, safety, communication in distributed systems, time-behavior). The analysis of domain concepts, development of middleware architectures, and comprehensive code generation were given prominence.

University of Duisburg-Essen – The Ruhr Institute for Software Technology (paluno)

Created in 2003 by the merger of the University of Duisburg and the University of Essen, the University of Duisburg-Essen is the youngest university in North Rhine-Westphalia and one of the ten largest universities in Germany. In many disciplines, the University of Duisburg-Essen ranks amongst the top ten German research universities. Over the past three years, research income has increased by approximately 100 percent.

In 2010, the research institute "paluno – The Ruhr Institute for Software Technology" at the University of Duisburg-Essen was founded. Paluno's six professors and their research teams bring in experience from application domains as diverse as insurance, automotive, healthcare, energy, and logistics. Their competencies cover most phases and layers of software engineering. With partners throughout Europe, paluno researches and applies methods and tools for design, implementation, and operation of future software-intensive systems. Paluno's research and transfer paradigm encourages mutual benefit from basic research, applied research, and bilateral industry cooperation.

In SPES 2020, the research team of Prof. Pohl, together with Bosch, led the work package ZP-AP2 on model-based requirements engineering. Paluno coordinated all activities in the area of requirements engineering within the central project and within the application projects. In addition, paluno co-led the research activities in the application domain "energy." paluno's main contribution is the development of a methodology for model-based requirements engineering of embedded systems.

In addition, researchers from paluno were members of the project architecture team that made major contributions towards an integrated model-based methodology.

University of Paderborn – Software Engineering Group

The Software Engineering Group of the University of Paderborn is headed by Prof. Dr. Wilhelm Schäfer. The main research topics are model-driven, component-based

development and analysis of software, including techniques based on UML (Unified Modeling Language). Embedded or mechatronic systems with real-time and safety-critical constraints, as well as business information systems, are considered as target domains. Further research areas include approaches for reengineering and the object-oriented specification of software process models.

The Software Engineering Group participates in several national and international research projects, often in close cooperation with partners from industry. The group is a founding member of the Software Quality Lab (s-lab), undertaking industry-driven research with a strong focus on software quality, and has a long tradition in cooperating with research groups from mechanical and electrical engineering of the Heinz Nixdorf Institute. In particular, the cooperation with the Product Engineering and Control Engineering Groups recently led to the formation of the Fraunhofer Project Group on Mechatronic Systems Design, which is located in Paderborn and belongs to the Fraunhofer-Institute for Production Technology IPT. Prof. Schäfer is a member of the board of directors of the project group and scientific director of its Software Engineering Department.

In SPES, the Software Engineering Group, together with s-lab and Fraunhofer IPT, participated in the automotive application project. In addition, Prof. Schäfer was deputy coordinator of this application project. In cooperation with the automotive supplier Hella KGaA Hueck & Co., the project developed a seamless model-based design methodology that complies with the maturity model Automotive SPICE. The methodology partially automates the transitions between the different design phases and viewpoints (from requirements to AUTOSAR) in a systematic way, with a strong focus on consistency and traceability. A further focus was on the integration of tools that allow the simulation of functional and real-time behavior in early development phases. In order to incorporate the results into the central project, the Software Engineering Group further contributed to the first three work packages of the central project.

E – List of Publications

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9, © Springer-Verlag Berlin Heidelberg 2012

A

[Althaus et al. 2011] E. Althaus, R. Naujoks, E. Thaden: A column generation approach to scheduling of periodic tasks, experimental algorithms. In: Proceedings of the 10th International Symposium, 2011.

[Arbeiter et al. 2010] C. Arbeiter, C. Gips, J. Wojtacki: Automatisierung des funktionalen Tests auf der Basis textuell-formalisierter Testspezifikationen, 3. Autotest, 2010.

[Arendt et al. 2011] Th. Arendt, S. Kranz, F. Mantz, N. Regnat, G. Taentzer: Towards syntactical model quality assurance in industrial software development: Process definition and tool support. In: Proceedings of Software Engineering 2011, 2011.

B

[Baumgart et al. 2011] A. Baumgart, E. Böde, M. Büker, W. Damm, G. Ehmen, Tayfun Gezgin, Stefan Henkler, Hardi Hungar, Bernhard Josko, Markus Oertel, Thomas Peikenkamp, Philipp Reinkemeier, Ingo Stierand, Raphael Weber: Architecture modeling. Technical Report. OFFIS, 2011.

[Botaschanjan and Hummel 2010] J. Botaschanjan, B. Hummel: Material flow abstraction of manufacturing systems. In: Proceedings of the International Conference on Theoretical Aspects of Computing, 2010.

[Botaschanjan and Harhurin 2009] J. Botaschanjan, A. Harhurin: Property-driven scenario integration. In: 7th IEEE International Conference on Software Engineering and Formal Methods, 2009.

[Broy et al. 2010] M. Broy, M. Feilkas, M. Herrmannsdoerfer, St. Merenda, D. Ratiu: Seamless model-based development: From isolated tools to integrated model engineering environments. In: Proceedings of the IEEE - Special Issue on Aerospace & Automotive, 2010.

[Buckl et al. 2010] Ch. Buckl, I. Gaponova, M. Geisinger, A. Knoll, E. Lee: Model-based specification of timing requirements. In: Proceedings of the 10th ACM international conference on Embedded software, 2010.

[Buckl et al. 2010] Ch. Buckl, D. Sojer, A. Knoll: FTOS: Model-driven development of fault-tolerant automation systems. In: Proceedings of the 15th IEEE International Conference on Emerging Techonologies and Factory Automation, 2010.

[Büker et al. 2011] M. Büker, W. Damm, G. Ehmen, A. Metzner, I. Stierand, E. Thaden: Automating the design flow for distributed embedded automotive applications: Keeping your time promises, and optimizing costs, too. In: Proceedings of the International Symposium on Industrial Embedded Systems, 2011.

[Büker et al. 2009] M. Büker, A. Metzner, I. Stierand: Testing real-time task networks with functional extensions using model-checking. In: Proceedings of the 14th International Conference on Emerging Technologies and Factory Automation, 2009.

C

[Campetelli et al. 2011] A. Campetelli, F. Hölzl, Ph. Neubeck: User-friendly model checking integration in model-based development. In: Proceedings of the 24th International Conference on Computer Applications in Industry and Engineering, 2011.

[Campetelli et al. 2010] A. Campetelli, M. V. Cengarle, I. Gaponova, A. Harhurin, D. Ratiu, J. Thyssen: Specification Techniques. Technical Report TUM-I1013, Technische Universität München. 2010.

[Ciolkowski 2009] M. Ciolkowski: What do we know about perspective-based reading? An approach for quantitative aggregation in software engineering. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, 2009.

[Clark et al. 2011] B. Clark, I. Stierand, E. Thaden: Cost-minimal pre-allocation of software tasks under real-time constraints. In: Proceedings of the 2011 Research in Applied Computation Symposium, 2011.

D

[Damm et al. 2011] W. Damm, H. Hungar, B. Josko, Th. Peikenkamp, I. Stierand: Using contract-based component specifications for virtual integration testing and architecture design. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, 2011.

[Domis and Trapp 2010] D. Domis, K. Höfig, M. Trapp: Consistency check algorithm for component-based refinements of fault trees. In: Proceedings of the International Symposium on Software Reliability Engineering, 2010.

[Domis and Trapp 2009] D. Domis, M. Trapp: Component-based abstraction in fault tree analysis. In: Proceedings of the International Conference on Computer Safety, Reliability and Security, 2009.

F

[Feilkas et al. 2009] M. Feilkas, A. Harhurin, J. Hartmann, D. Ratiu, W. Schwitzer: Motivation and introduction of a system of abstraction layers for embedded systems. Technical Report, TUM-I0925. Technische Universität München. 2009.

[Fieber et al. 2009] F. Fieber, N. Regnat, B. Rumpe: Assessing usability of model driven development in industrial projects. In: Proceedings of the 4th Workshop "From code centric to model centric software engineering: Practices, Implications and ROI", 2009.

[Foehr et al. 2011] M. Foehr, A. Lüder, T. Jäger, A. Fay, T. Wagner: Development of a method to analyze the impact of manufacturing systems engineering on product quality. In: Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation, 2011.

G

[Gellermann et al. 2012] A. Gellermann, T. Jäger, A. Fay, Th. Wagner, A. Müller-Martin: Analyse und Optimierung von Engineering-Schnittstellen. In: Proceedings of Automation, 2012.

[Gezgin et al. 2011] T. Gezgin, R. Weber, M. Girod: A refinement checking technique for contract-based architecture designs. In: Proceedings of the 4th International Workshop on Model Based Architecting and Construction of Embedded Systems, 2011.

[Groß et al. 2010] A. Groß, J. Dörr, I. Menzel, M. Müller: Experimenteller Vergleich zweier Techniken zur Anforderungsspezifikation: Use Cases vs. Funktionale Spezifikation. In: Softwaretechnik-Trends, 2010.

[Groß et al. 2010] A. Groß, J. Dörr, I. Menzel, M. Müller: Experiment package: An experimental comparision regarding the completeness of functional requirements specifications. Technical Report No. 040.10/E., Fraunhofer IESE, 2010.

[Groß et al. 2009] A. Groß, J. Dörr, I. Menzel, M. Müller: Use Cases vs. Funktionale Spezifikation: Ein experimenteller Vergleich zweier Techniken zur Anforderungs-spezifikation, GI-Fachgruppen-Treffen Requirements Engineering, 2009.

[Guzmán et al. 2009] L. Guzmán, J. Münch, D. Rombach: Qualitative synthesis of evidence in software engineering – A systematic review. Technical Report, Fraunhofer IESE, 2009.

H

[Harhurin et al. 2009] A. Harhurin, J. Hartmann, D. Ratiu: Motivation and formal foundations of a comprehensive modeling theory for embedded systems. Technical Report, TUM-I0924. Technische Universität München, 2009.

[Heuer et al. 2010] A. Heuer, C.J. Budnik, S. Konrad, K. Lauenroth, K., Pohl: Formal definition of syntax and semantics for documenting variability in activity diagrams. In: Proceedings of the 14th International Software Product Line Conference, 2010.

[Herrmannsdoerfer et al. 2011] M. Herrmannsdoerfer, D. Ratiu, M. Koegel: Metamodel usage analysis for identifying metamodel improvements. In: Software Language Engineering, 2011.

[Herrmannsdoerfer et al. 2011] M. Herrmannsdoerfer, S. Vermolen, G. Wachsmuth: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Software Language Engineering, 2011.

[Herrmannsdoerfer 2011] M. Herrmannsdoerfer. COPE - A workbench for the coupled evolution of metamodels and models. In: Software Language Engineering, 2011.

[Herrmannsdoerfer et al. 2010] M. Herrmannsdoerfer, D. Ratiu, G. Wachsmuth: Language evolution in practice: The history of GMF. In: Proceedings of the 2nd International Conference on Software Language Engineering, 2010.

[Herrmannsdoerfer et al. 2010] M. Herrmannsdoerfer, Th. Kofler, S. Merenda, D. Ratiu, J. Thyssen: Model-based development tools for embedded systems in the industry – Results from an empirical investigation. In: Proceedings of ENVISION 2020, 2010.

[Herrmannsdoerfer and Ratiu 2010] M. Herrmannsdoerfer, D. Ratiu: Limitations of automating model migration in response to metamodel adaptation. In: Proceedings of the Joint ModSE-MCCM Workshop on Models and Evolution, 2010.

[Herrmannsdoerfer and Koegel 2010] M. Herrmannsdoerfer, M. Koegel: Towards a generic operation recorder for model evolution. In: Proceedings of the 1st International Workshop on Model Comparison in Practice, 2010.

[Herrmannsdoerfer and Merenda 2009] M. Herrmannsdoerfer, S. Merenda: Result of the tool questionnaire. Technical Report TUM-I0929. Technische Universität München, 2009.

[Hilbrich 2011] R. Hilbrich 2011. Planung sicherheitskritischer Systeme – Tool schreibt Ablaufpläne für Multi-Core-Systeme. Innovisions, 2011.

[Hilbrich et al. 2011] R. Hilbrich, J. R. van Kampenhout, H.-J. Goltz. Modellbasierte Generierung von statischen Schedules für sicherheitskritische, eingebettete Systeme mit Multicore Prozessoren und harten Echtzeitanforderungen. In: Proceedings oft he Workshop Echtzeit, 2011.

[Hilbrich and van Kampenhout 2011] R. Hilbrich und J. R. van Kampenhout. Partitioning and task transfer on NoC-based many-core processors in the avionics domain. In: Softwaretechnik Trends, 2011.

[Hilbrich and Goltz 2011] R. Hilbrich, H.-J. Goltz: Model-based generation of static schedules for safety critical multi-core systems in the avionics domain, In: Proceedings of the 4th international Workshop on Multicore Software Engineering, 2011.

[Hilbrich and Svacina 2010] R. Hilbrich, J. Svacina: Verifizierung statischer Schedules für die Zertifizierung. In: Proceedings of the Embedded Software Engineering Kongress, 2010.

[Hilbrich and van Kampenhout 2010] R. Hilbrich, J. R. van Kampenhout: Dynamic reconfiguration in NoC-based MPSoCs in the avionics domain. In: Proceedings of the 3rd international Workshop on Multicore Software Engineering, 2010.

[Höfig 2011] K. Höfig: T.iming overhead analysis for fault tolerance mechanisms. In: Workshopband Software Engineering, Lecture Notes in Informatics, 2011.

[Höfig and Domis 2011] K. Höfig, D. Domis: Failure-dependent execution time analysis. In: Proceedings of ISARCS, 2011.

[Höfig 2011] K. Höfig: FDTA - A tool chain for failure dependent timing analysis. In: Proceedings of WCET 2011.

[Holtmann et al. 2011] J. Holtmann, J. Meyer, M. Meyer: A seamless model-based development process for automotive systems. In: Workshopband Software Engineering, Lecture Notes in Informatics, 2011.

[Holtmann et al. 2011] J. Holtmann, J. Meyer, M. von Detten: Automatic validation and correction of formalized, textual requirements. In: Proceedings of the Fourth IEEE International Conference on Software Testing, Verification and Validation Workshops, 2011.

[Holtmann 2010] J. Holtmann: Mit Satzmustern von textuellen Anforderungen zu Modellen. In: OBJEKTspektrum Online Themenspecial Requirements Engineering, 2010.

[Holtmann et al. 2010] J. Holtmann, J. Meyer, W. Schäfer, U. Nickel: Eine erweiterte Systemmodellierung zur Entwicklung von softwareintensiven Anwendungen in der Automobilindustrie. In: Workshopband Software Engineering, Lecture Notes in Informatics, 2010.

[Hungar 2011] H. Hungar: Compositionality with strong assumptions. In: Proceedings of the Nordic Workshop on Programming Theory, 2011.

J

[Jäger et al. 2012] T. Jäger, A. Fay, Th. Wagner, U. Löwen: Comparison of engineering results within domain specific languages regarding information contents and intersections. In: Proceedings of 9th International Multi-Conference on Systems, Signals and Devices, 2012.

[Jäger et al. 2011] T. Jäger, A. Fay, Th. Wagner: Systematische Erfassung und Bewertung von gewerkeübergreifenden Schnittstellen in Engineering-Workflows. In: Proceedings oft he 8th Symposium Informationstechnologien für Entwicklung und Produktion in der Verfahrenstechnik, 2011.

[Jäger et al. 2011] T. Jäger, A. Fay, H. Figalist, Th. Wagner: Systematische Risikominimierung im Engineering mit Abhängigkeitsanalyse und Schlüsseldokumenten Vorgehen und Ergebnisse einer Fallstudie zur Erfassung der gewerkeübergreifenden Informationsschnittmenge im Engineering automatisierter Anlagen. In: Proceedings of Automation, 2011.

[Jäger et al. 2011] T. Jäger, A. Fay, T. Wagner, U. Löwen: Mining technical dependencies throughout engineering process knowledge. In: Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation, 2011.

K

[Kagel and Lim 2009] S. Kagel, M. Lim: Herausforderungen der Variantenentwicklung im Anforderungsmanagement meistern. GI-Fachgruppentreffen Requirements Engineering, 2009.

[Koegel et al. 2010] M. Koegel, M. Herrmannsdoerfer, Y. Liz, J. Helming, J. David: Comparing state- and operation-based change tracking on models. In: Proceedings of Enterprise Distributed Object Computing Conference, 2010.

[Koegel et al. 2010] M. Koegel, H. Naughton, J. Helming, M. Herrmannsdoerfer: Collaborative model merging. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 2010.

[Koegel et al. 2010] M. Koegel, M. Herrmannsdoerfer, O. von Wesendonk, J. Helming: Operation-based conflict detection. In: Proceedings of the 1st International Workshop on Model Comparison in Practice, 2010.

[Kofler and Ratiu 2010] Th. Kofler, D. Ratiu. Towards a reusable unified basis for representing business domain knowledge and development artifacts in systems engineering. In: Proceedings of the Workshop on Domain Engineering, Advances in Conceptual Modeling – Applications and Challenges, 2010.

L

[Lackner and Tahirbegovic 2011] H. Lackner, S. Tahirbegovic: Nicht kapitulieren, sondern automatisieren. In: Medizin & Technik 5, 2011, pp. 24-25.

[Lackner et al. 2009] H. Lackner, J. Svacina, H. Schlingloff: Test case generation from workflow-based requirement specifications. In: Proceedings of the 2009 Workshop on Concurrency, Specification and Programming, 2009.

[Lauenroth and Pohl 2009] K. Lauenroth, K. Pohl: Model checking of domain Artifacts in product line engineering. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering, 2009.

[Lim 2011] M. Lim: Qualitätssicherung medizinischer Software - Durchgängig automatisiert testen. In: Medizin+Elektronik 2, 2011, pp. 39-41.

[Lim and Loose 2010] M. Lim, M. Loose: Mit Varianten-Management erfolgreich zum Ziel. In: Elektronik Automotive 1, 2010, pp. 33-35.

[Löwen and Wagner 2011] U. Löwen, Th. Wagner: Analyse von Engineering-Workflows als Basis für den optimalen Einsatz von Engineering-Werkzeugen. In: Proceedings of the 8th Symposium Informationstechnologien für Entwicklung und Produktion in der Verfahrenstechnik, 2011.

[Löwen and Wagner 2009] U. Löwen, Th. Wagner: Modellierung komplexer technischer Systeme - Anforderungen und Erfahrungen aus dem Anlagenbau. In: Proceedings of Mechatronik, 2009.

[Löwen et al. 2009] U. Löwen, K. Dencovski, Th. Wagner: Integration of information and tools: Where are the gaps? In: Proceedings of the 15th Daratech Plant Conference, 2009.

[Lüder et al. 2010] A. Lüder, L. Hundt, M. Foehr, Th. Wagner, J.-J. Zaddach: Manufacturing system engineering with mechatronical units. In: Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, 2010.

M

[Mattheis et al. 2012] S. Mattheis, T. Schuele, A. Raabe, Th. Henties, U. Gleim: Work stealing strategies for parallel stream processing in soft real-time systems. In: Proceedings of the International Conference on Architecture of Computing Systems, 2012.

[Menzel et al. 2010] I. Menzel, M. Müller, A. Groß, J. Dörr: An experimental comparison regarding the completeness of functional requirements specifications. In: Proceedings of the 18th IEEE international Requirements Engineering Conference, 2010.

[Meyer and Holtmann 2011] J. Meyer, J. Holtmann: Eine durchgängige Entwicklungsmethode von der Systemarchitektur bis zur Softwarearchitektur mit AUTOSAR. In: Proceedings of the Workshop on Modellbasierte Entwicklung eingebetteter Systeme VII, 2011.

[Meyer et al. 2011] J. Meyer, J. Holtmann, M. Meyer: Formalisierung von Anforderungen und Betriebssystemeigenschaften zur frühzeitigen Simulation von eingebetteten, automobilen Systemen. In: Proceedings of the 8th Paderborner Workshop Entwuf mechatronischer Systeme, 2011

N

[Nickel et al. 2010] U. Nickel, J. Meyer, T. Kramer: Wie hoch ist die Performance? In: Automobil-Elektronik 3, 2010, pp. 36-38.

P

[Pohl and Sikora 2009] K. Pohl, E. Sikora: COSMOD-RE - Verzahnung des Architekturentwurfs mit dem Requirements Engineering. In: OBJEKTspektrum, Online Themenspecial Architekturen, 2009.

[Post et al. 2011] A. Post, I. Menzel, I., A. Podelski: Applying restricted English grammar on automotive requirements - does it work? A case study. In: Proceedings of the 17th International Working Conference on Requirements Engineering - Foundation for Software Quality, 2011.

R

[Ratiu et al. 2009] D. Ratiu, J. Thyssen, W. Schwitzer: A system of abstraction layers for the seamless development of embedded software systems. Technical Report, TUM-I0928. Technische Universität München, 2009.

[Reinkemeier et al. 2011] Ph. Reinkemeier, I. Stierand, Ph. Rehkop, S. Henkler: A pattern-based requirement specification language - mapping automotive specific timing requirements. In: Proceedings of ENVISION 2020, 2011.

[Rose et al. 2010] L. M. Rose, M. Herrmannsdoerfer, J. R. Williams, D. S. Kolovos, K. Garces, R. F. Paige, F. A. C. Polack: A comparison of model migration tools. In: Model Driven Engineering Languages and Systems, 2010.

[Rüth 2009] C. Rüth.: Intelligente Maschinen ohne Denkfehler. Faszination Forschung 1, 2009, pp. 50-57.

S

[Sadeghipour and Wiesbrock 2011] S. Sadeghipour, H.-W. Wiesbrock: Systematische Datenüberdeckung im Modellzentrierten Test. Proceedings of the 4th Symposium Testen im System- und Software-Life-Cycle, 2011.

[Sadeghipour and Wiesbrock 2011] S. Sadeghipour, H.-W. Wiesbrock: Auswertung automatisch generierter Testfälle durch regelbasierte Testüberwachung. In: Proceedings of the Embedded Software Engineering Kongress, 2011.

[Sadeghipour 2010] S. Sadeghipour: Testautomatisierung: Ein akademisches Thema? In: Proceedings on the Advances in Testing: Academia meets Industry, 2010.

[Schneider and Trapp 2010] D. Schneider, M. Trapp: Conditional safety certificates in open systems. In: Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety, 2010

[Schlingloff 2011] H. Schlingloff: Entwicklerhilfe für Eingebettete Systeme. Interview, 2011.

[Schüle 2011] T. Schüle: Efficient parallel execution of streaming applications on multi-core processors. In: Proceedings of the International Conference on Parallel, Distributed and Network-Based Computing, 2011.

[Schüle 2009] T. Schüle: A coordination language for programming embedded multi-core systems. In: Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies, 2009.

[Siemens CT 2011] Siemens CT: Simulation Based Engineering - frühzeitige Validierung von Anlagenkonzepten. In: Proceedings of the 8th Paderborner Workshop Entwurf mechatronischer Systeme, 2011.

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal 17(1), 2012, pp. 57-78.

[Sikora et al. 2011] E. Sikora, B. Tenbergen, K. Pohl: Requirements engineering - An investigation of industry needs. In: Proceedings of the 17th International Working Conference on Requirements Engineering - Foundation for Software Quality, 2011.

[Sikora et al. 2010] E. Sikora, M. Daun, K. Pohl: Supporting the consistent specification of scenarios across multiple abstraction levels. In: Proceedings of the 16th Intermatonal Working Conference on Requirements Engineering - Foundation for Software Quality, 2010.

[Sikora and Pohl 2010] E. Sikora, K. Pohl: Evaluation eines modellbasierten Requirements-Engineering-Ansatzes für den Einsatz in der Motorsteuerungs-Domäne. In: Proceedings of ENVISION 2020, 2010.

[Sikora et al. 2010] E. Sikora, B. Tenbergen, K. Pohl: Modellbasiertes Requirements Engineering - Eine Situationsanalyse zum Stand der Praxis. In: Softwaretechnik-Trends 30(1), 2010.

[Sojer et al. 2012] D. Sojer, Ch. Buckl, A. Knoll: Deriving fault-detection mechanisms from safety requirements. In: Computer Science - Research and Development, 2012.

[Sojer 2011] D. Sojer: Synthesis of fault detection mechanisms. In Proceedings of the 35th IEEE International Computer Software and Applications Conference, 2011.

[Sojer et al. 2011] D. Sojer, Ch. Buckl, A. Knoll: Synthesis of diagnostic techniques based on an IEC 61508-aware metamodel. In: Proceedings of the 6th Symposium on Industrial Embedded Systems, 2011.

[Sojer et al. 2010] D. Sojer, Ch. Buckl, A. Knoll: Vom Modell zum Code für IEC 61508, ISO 26262 und Co. In: Proceedings of the 3rd Embedded Software Engineering Congress, 2010.

[Sojer et al. 2010] D. Sojer, Ch. Buckl, A. Knoll: Propagation, transformation and refinement of safety requirements. In: Proceedings of the 3rd Workshop on Non-functional System Properties in Domain Specific Modeling Languages, 2010.

[Sojer et al. 2010] D. Sojer, Ch. Buckl, A. Knoll. Formal modeling of safety requirements in the model-driven development of safety critical embedded systems. In: Proceedings of the Eighth European Dependable Computing Conference, 2010.

[Sojer et al. 2010] D. Sojer, Ch. Buckl, A. Knoll. Stand und Anforderungen an eine Werkzeugunterstuetzung zur Entwicklung von Automatisierungssoftware. Technical Report TUM-I1003, Technische Universität München, 2010.

[Stallbaum and Rzepka 2010] H. Stallbaum, M. Rzepka: Toward DO-178B-compliant Test Models. In: Proceedings of the 7th Workshop on Model-Driven Engineering, Verification and Validation, 2010.

[Stallbaum et al. 2010] H. Stallbaum, A. Metzger, K. Pohl: Der Einsatz quantitativer Sicherheitsanalysen für den risikobasierten Test eingebetteter Systeme. In: Proceedings of Software Engineering, 2010.

[Strube et al. 2011] M. Strube, A. Fay, S. Truchat, H. Figalist: Funktionale Anlagenbeschreibung als Basis der Modernisierungsplanung. In: Proceedings of Automation, 2011.

[Strube et al. 2011] M. Strube, T. Jäger, A. Fay: Integriertes Engineering durch Zusammenführen von Prozess- und Anlagenbeschreibung – Ein Konzept zur ganzheitlichen Beschreibung von Produktionsanlagen. In: Proceedings of the 14th Conference IFF-Wissenschaftstage, 2011.

[Strube et al. 2011] M. Strube, S. Runde, A. Fay, H. Figalist: Risk Minimization in Modernization Projects of Plant Automation – a Knowledge-Based Approach by means of Semantic Web Technologies. In: Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'2011, September 5-9, 2011 in Toulouse, France, ISBN: 978-1-4577-0016-3, 2011.

[Strube et al. 2011] M. Strube, A. Fay, S. Truchat, H. Figalist: Modellgestützte Modernisierungsplanung. In: Automatisierungstechnische Praxis 8, 2011, pp. 889-895.

[Strube and Fay 2010] M. Strube, A. Fay: Brückenschlag zwischen Prozess- und Anlagenbeschreibung. In: Automatisierungstechnische Praxis 9, 2010, pp. 26-27.

[Strube et al. 2010] M. Strube, A. Fay, S. Truchat, H. Figalist: Funktionale Anlagenbeschreibung als Basis der Modernisierungsplanung. In: Proceedings of Automation, 2010.

T

[Tahirbegovic 2010] S. Tahirbegovic: Herzschlag übers iPhone. In: Medizin und Technik: 4, 2010.

[Tetzner and Gewald 2009] T. Tetzner, N. Gewald: Mechatronisches Konzept im Engineering von Industrieanlagen – Anforderungen aus Anwendersicht. In: Proceedings of the 6th Berlin-Aachener Symposium, 2009.

[Thaden et al. 2010] E. Thaden, H. Lipskoch, A. Metzner, I. Stierand: Exploiting gaps in fixed-priority preemptive schedules for task insertion. In: Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2010.

[Thyssen et al. 2010] J. Thyssen, D. Ratiu, W. Schwitzer, A. Harhurin, M. Feilkas, E. Thaden: A system for seamless abstraction layers for model-based fevelopment of embedded software. In: Proceedings of ENVISION 2020, 2010.

W

[Wagner and Löwen 2010] Th. Wagner, U. Löwen: Modellierung: Grundlage für integriertes Engineering. In: Proceedings of Automation, 2010.

[Wehrstedt et al. 2011] J. Ch. Wehrstedt, R. Rosen, A. Pirsing, C. Dietz: Simulation Based Engineering-Frühzeitige Validierung von Anlagekonzepten. In Proceedings of the 8th Paderborner Workshop Entwurf mechatronischer Systeme, 2011.

[Weißleder and Lackner 2010] St. Weißleder, H. Lackner: System models vs. test models - Distinguishing the undistinguishable? In: Proceedings of the MoTes2010 Workshop, 2010.

[Weyer 2011] Th. Weyer: Kohärenzprüfung von Anforderungsspezifikationen - Ein Ansatz zur Prüfung der Kohärenz von Verhaltensspezifikationen gegen Eigenschaften des operationellen Kontexts, Südwestdeutscher Verlag für Hochschulschriften, 2011.

[Wiesbrock 2011] H.-W. Wiesbrock: Stochastische Robustheitstests von automobilen Steuergeräten und ihre automatische Auswertung. In: Proceedings of Embedded goes Medical - Advances in Testing: Academia meets Industry, 2011.

[Wiesbrock 2010] H.-W. Wiesbrock: Ansätze zum Nachweis der Gleichwertigkeit von Software-Komponenten. In: Proceedings of Envision 2020, 2010.

[Wiesbrock 2010] H.-W. Wiesbrock: Rezertifizierung von Software-Komponenten - Ansätze zum Nachweise ihrer Gleichwertigkeit In: Proceedings of Embedded goes Medical - Advances in Testing: Academia meets Industry, 2010.

[Wiesbrock 2009] H.-W. Wiesbrock: Das Entwicklungsdreieck Anforderungen – Design – Test: Ein Praxisbericht über die Kopplung der einzelnen Artefakte. In: Proceedings of Software-Quality-Days, 2009.

Z

[Zimmer et al. 2011] B. Zimmer, S. Bürklen, M. Knoop, J. Höfflinger, M. Trapp: Vertical safety interfaces - Improving the efficiency of modular certification. In: Proceedings of the 30th International Conference of Computer Safety, Reliability, and Security, 2011.

F – Index

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9, © Springer-Verlag Berlin Heidelberg 2012

F

G

H

I

L

M

O

P

Q

R

S

T

V