Wolfgang Böhm
Manfred Broy
Cornel Klein
Klaus Pohl
Bernhard Rumpe
Sebastian Schröck *Eds.*

Model-Based Engineering of Collaborative Embedded Systems

Extensions of the SPES Methodology

OPEN ACCESS

Springer

www.dbooks.org

Model-Based Engineering of Collaborative Embedded Systems

Wolfgang Böhm • Manfred Broy
Cornel Klein • Klaus Pohl
Bernhard Rumpe • Sebastian Schröck
Editors

Model-Based Engineering of Collaborative Embedded Systems

Extensions of the SPES Methodology

Springer

www.dbooks.org

Editors
Wolfgang Böhm
Fakultät für Informatik
Technische Universität München
Garching, Germany

Manfred Broy
Fakultät für Informatik
Technische Universität München
Garching, Germany

Cornel Klein
Corporate Technology, CT RDA SSI
Siemens AG
München, Germany

Klaus Pohl
Software Systems Engineering
University of Duisburg-Essen
Essen, Germany

Bernhard Rumpe
Lehrstuhl Software Engineering
RWTH Aachen University
Aachen, Germany

Sebastian Schröck
Assembly Technique (CR/APA1)
Robert Bosch (Germany)
Renningen, Germany



This book is an open access publication.
ISBN 978-3-030-62135-3 ISBN 978-3-030-62136-0 (eBook)
https://doi.org/10.1007/978-3-030-62136-0

© The Editor(s) (if applicable) and The Author(s) 2021
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Collaborative embedded systems (CESs)

With the transition from classical embedded systems to networked, collaborative embedded systems (CESs), a wide range of new applications is emerging. The ability of companies to efficiently develop CESs of the highest quality is therefore a decisive competitive factor. However, collaboration means a leap in complexity. In addition to the quality of the embedded system, collaborative networks that change dynamically at runtime must also be considered. The product success of systems with embedded software is today even more determined by software quality. Therefore, it is essential to master the complexity of CESs with efficient and effective methods.

As more and more domains are becoming increasingly digitalized, technologies for software development are becoming more and more heterogeneous. This makes it even more important for research on software engineering in particular to provide generalized, generally applicable methods and techniques for the various types of software in order to provide a solid basis for growing diversification.

Methodologically sound approach

The modeling of systems—and here, the explicit modeling of structures, behavior, interaction patterns, dynamics, functional constraints, and non-functional constraints—plays an essential role in a methodologically sound approach to software and system development.

SPES methodology

In the funded projects "Software Platform Embedded Systems" (SPES2020) and the follow-up project SPES_XT, the foundations for a comprehensive methodological toolkit for the integrated model-based development of embedded systems were developed. The methods and tools developed in these projects allow the complexity of embedded systems to be mastered in the development process.

www.dbooks.org

Therefore, the results of the SPES projects provide an excellent starting point for tackling the next level of complexity, which is reached with the development of CESs.

CrESt project

The project "Collaborative Embedded Systems" (CrESt), funded by the German Federal Ministry of Education and Research (BMBF)[1], is the successor of the two SPES projects. It aimed at adapting and complementing the methodology developed and the underlying modeling techniques to the challenges of dynamic and dynamically changing structures of CESs based on the SPES development methodology. Further information on the CrESt project is available on its website[2], which also features selected project deliverables.

New development challenges

In order to cope with the high complexity of the individual systems and the dynamically formed interaction structures at runtime, which are partly based on uncertain context information, highly advanced, powerful development methods are required that extend the current state of the art in the development of embedded systems and cyber-physical systems. The development of CESs goes hand in hand with important safety and security issues. Our case studies are therefore selected from areas that are highly relevant for Germany's economy (automotive, industrial production, power generation, and robotics). Given its focus, the project also supports the digitalization of complex and transformable industrial plants in the context of the German government's "Industry 4.0" initiative. In addition, the expected project results provide a solid foundation to the mosaic for implementing the German government's high-tech strategy "Innovations for Germany" in the coming years.

Contributions

The methodological contributions of the project support the effective and efficient development of CESs in dynamic and uncertain contexts, with special emphasis on the reliability and variability of individual systems and the creation of networks of such systems at runtime. The manifold potentials of such systems are expected to have a sustainable positive impact on the information society. In many leading branches of German high technology, such as the automotive industry, automation, industrial plant engineering, as well as health, logistics, mobility, medicine, and in the service sector, CESs will play a central role in the future — for example, in order to reliably and completely automate tasks in highly complex dynamic everyday situations and to comprehensively support people in such dynamically developing networks of systems of systems.

[1] Funded by the BMBF under grant number 01IS16043

[2] CrESt project website: www.crest.in.tum.de

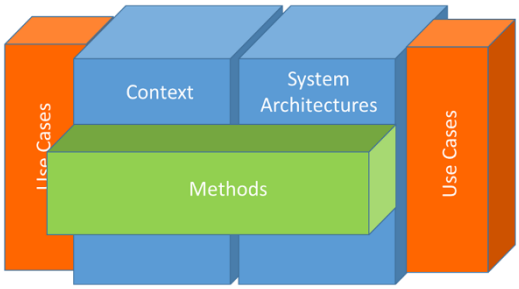


Fig. 1: *CrESt project structure*

Project structure

In the CrESt project, 22 partners from industry and science have joined forces to research and develop new techniques and methods for the development of CESs based on the results of the SPES projects. The work was conducted along a three-dimensional project structure, whose central elements were the engineering challenges that have been the subject of work across domains. Orthogonally to this, work on cross-domain topics aimed at promoting the integrability and consistency of the solutions developed. Finally, in the third dimension, the specifics of the different application domains were considered and evaluated based on case studies. This allowed the methodological work between the different engineering challenges to be shared and thus significant synergies to be released.

The present book describes core results of the CrESt project and is supplemented by the classification of our results in the context of "model-based systems engineering" (MBSE). The relevance has become clear at least since the discussion about Industry 4.0 and cyber-physical systems (CPS).

Acknowledgements

Many people have made substantial contributions to this book. First of all, we would like to thank the project partners and their employees, who have contributed with great dedication to the development of the project results.

Secondly, we would like to thank the steering committee of the CrESt project for their continuous guidance and support throughout the project and for encouraging us to document major project results in this book.

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

Special thanks are also due to the many reviewers of the individual chapters of the book, who have contributed significantly to improve the quality of the individual chapters.

The results presented in this book have been made possible through the funding of the BMBF; in particular, we would like to thank

www.dbooks.org

Dr. Ute Bernhard and Dr. Michael Weber, as well as Mr. Dirk Günther from the project management agency of the German Aerospace Center (DLR) for their continuous support.

Last but not least, we would like to express our deepest thanks to Dr. Andreas Wortmann for the excellent management of the overall book production process.

We hope that you will enjoy reading the book and using the knowledge presented in your daily business.

Wolfgang Böhm
Manfred Broy
Cornel Klein
Klaus Pohl
Bernhard Rumpe
Sebastian Schröck

Summer 2020

Table of Contents

1 Use Cases 1
1.1 Introduction 2
1.2 Vehicle Platooning 2
1.3 Adaptable and Flexible Factory 6
1.4 Autonomous Transport Robots 10
2 Engineering of Collaborative Embedded Systems 15
2.1 Introduction 16
2.2 Background 16
2.3 Collaborating Embedded Systems 19
2.4 Problem Dimensions of Collaborative Embedded Systems 26
2.5 Application in the Domains “Cooperative Vehicle Automation” and “Industry 4.0” 29
2.6 Concepts and Methods for the Development of Collaborative Embedded Systems 37
2.7 Conclusion 45
2.8 Literature 46
2.9 Appendix 48
3 Architectures for Flexible Collaborative Systems 49
3.1 Introduction 50
3.2 Designing Reference Architectures 50
3.3 Reference Architecture for Operator Assistance Systems 57
3.4 Checkable Safety Cases for Architecture Design 62
3.5 Conclusion 68
3.6 Literature 69
4 Function Modeling for Collaborative Embedded Systems 71
4.1 Introduction 72
4.2 Methodological Approach 73
4.3 Background 75
4.4 Metamodel for Functions of CESs and CSGs 75
4.5 Evaluation of the Metamodel 82
4.6 Application of the Metamodel 85
4.7 Related Work 89

www.dbooks.org

4.8 Conclusion 90
4.9 Literature 91
5 Architectures for Dynamically Coupled Systems 95
5.1 Introduction 96
5.2 Specification Modeling of the Behavior of Collaborative System Groups 98
5.3 Modeling CES Functional Architectures 103
5.4 Extraction of Dynamic Architectures 108
5.5 Functional Safety Analysis (Online) 117
5.6 Conclusion 121
5.7 Literature 121
6 Modeling and Analyzing Context-Sensitive Changes during Runtime ... 125
6.1 Introduction and Motivation 126
6.2 Solution Concept 126
6.3 Ontology and Modeling 127
6.4 Model Integration and Execution 138
6.5 Conclusion 143
6.6 Literature 144
7 Handling Uncertainty in Collaborative Embedded Systems Engineering 147
7.1 Uncertainty in Collaborative Embedded Systems 148
7.2 Modeling Uncertainty 151
7.3 Analyzing Uncertainty 160
7.4 Conclusion 168
7.5 Literature 169
8 Dynamic Safety Certification for Collaborative Embedded Systems at Runtime 171
8.1 Introduction and Motivation 172
8.2 Overview of the Proposed Safety Certification Concept 173
8.3 Assuring Runtime Safety Based of Modular Safety Cases 174
8.4 Design and Runtime Contracts 188
8.5 Conclusion 194
8.6 Literature 194
9 Goal-Based Strategy Exploration 197
9.1 Introduction 198
9.2 Goal Modeling for Collaborative System Groups 198

9.3 Goal-Based Strategy Development 201
9.4 Goal Operationalization (KPI Development) 205
9.5 Modeling Methodology for Adaptive Systems with MATLAB/Simulink ... 207
9.6 Collaboration Framework for Goal-Based Strategies 210
9.7 Conclusion 214
9.8 Literature 215
10 Creating Trust in Collaborative Embedded Systems 217
10.1 Introduction 218
10.2 Building Trust during Design Time 219
10.3 Building Trust during Runtime 225
10.4 Monitoring Collaborative Embedded Systems 228
10.5 Conclusion 236
10.6 Literature 237
11 Language Engineering for Heterogeneous Collaborative Embedded Systems 239
11.1 Introduction 240
11.2 MontiCore 242
11.3 Language Components 243
11.4 Language Component Composition 246
11.5 Language Product Lines 248
11.6 Conclusion 251
11.7 Literature 251
12 Development and Evaluation of Collaborative Systems using Simulation 255
12.1 Introduction 256
12.2 Challenges in Simulating Collaborative Embedded Systems 258
12.3 Simulation Methods 262
12.4 Application 266
12.5 Conclusion 267
12.6 Literature 267
13 Tool Support for Co-Simulation-Based Analysis 269
13.1 Introduction 270
13.2 Interaction of Different Simulations 270
13.3 General Tool Architecture 275

www.dbooks.org

13.4 Implementing Interoperability for Co-Simulation 276
13.5 Distributed Co-Simulation 278
13.6 Analysis of Simulation Results 280
13.7 Conclusion 280
13.8 Literature 281
14 Supporting the Creation of Digital Twins for CESs 283
14.1 Introduction 284
14.2 Building Trust through Digital Twin Evaluation 285
14.3 Conclusion 292
14.4 Literature 293
15 Online Experiment-Driven Learning and Adaption 295
15.1 Introduction 296
15.2 A Self-Optimization Approach for CESs 297
15.3 Illustration of CrowdNav 300
15.4 Conclusion 302
15.5 Literature 303
16 Compositional Verification using Model Checking and Theorem Proving 305
16.1 Introduction 306
16.2 Approach 307
16.3 Example 308
16.4 Conclusion 313
16.5 Literature 313
17 Artifact-Based Analysis for the Development of Collaborative Embedded Systems 315
17.1 Introduction 316
17.2 Foundations 317
17.3 Artifact-Based Analysis 319
17.4 Artifact Model for Systems Engineering Projects with Doors NG and Enterprise Architect 325
17.5 Conclusion 329
17.6 Literature 330
18 Variant and Product Line Co-Evolution 333
18.1 Introduction 334

18.2 Product Line Engineering 336
18.3 Propagating Updates from Domain Engineering Level to Application Engineering Level 337
18.4 Propagating Changes from Application Engineering Level to Domain Engineering Level 344
18.5 Conclusion 350
18.6 Literature 350
19 Advanced Systems Engineering 353
19.1 Introduction 354
19.2 Advanced Systems Engineering 355
19.3 MBSE as an Essential Basis 356
19.4 The Integrated Approach of SPES and SPES_XT 358
19.5 Methodological Extensions: From SPES to ASE 361
19.6 Conclusion 363
19.7 Literature 364

Appendices 365
A - Author Index 367
B - Partner 375
C - List of Publications 391

www.dbooks.org

Check for updates

Holger Schlingloff, Fraunhofer FOKUS

1

CrESt Use Cases

In this chapter, we present three use cases that are used throughout this book to demonstrate the various systems engineering methods presented: vehicle platooning, adaptable and flexible factories, and autonomous transport robots. The use cases are chosen from real-life industrial tasks and exhibit all software engineering challenges that are specific to the development of collaborative embedded systems.

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_1

1.1 Introduction

To derive and present the systems engineering methods described in this volume, three different industrial use cases are used throughout the book. These are vehicle platooning, adaptable and flexible factories, and autonomous transport robots. In the following, we describe each use case up to a level of detail that shows clearly how the respective process building blocks contribute to the overall development of the use case. For each use case, we first give some remarks on the historical evolution of the domain, then describe requirements and application scenarios for the use case, and finally describe the main challenges for development to be addressed in the rest of the book.

1.2 Vehicle Platooning



In the "Vehicle Platooning" use case, we consider a group of vehicles that share the goal of traveling together at high speed for some distance. With the vehicles driving in a low-distance formation, the overall air resistance is decreased and fuel consumption is significantly reduced. Furthermore, more vehicles fit onto the street and traffic may be more efficient. However, in order to avoid crashing into one another, the vehicles have to communicate constantly. Scenarios within this use case are as follows: forming and dissolving a platoon, as well as single vehicles joining and leaving a platoon.

Cruise control (CC) in cars has been known since the 1950s. Up to now, such systems have been and still are limited to isolated control decisions executed individually based on local sensor data. In the future, vehicle-to-vehicle and vehicle-to-roadside communication technology will enable the cruise control systems to consider a vast range of additional context information (e.g., general traffic conditions, dangerous situations ahead, etc.). This will enable the cruise control system to establish effective collaboration between vehicles. This kind of collaborative cruise control will be the central component of upcoming fully autonomous vehicles.

Adaptive cruise control (ACC) is a step towards such a collaborative cruise control. It is an enhancement of conventional cruise control systems that allows the vehicle equipped with ACC to follow a vehicle in front with a pre-selected time gap by controlling the engine, power train, and/or service brakes. This means that the

www.dbooks.org

ACC is a system that requests the onboard computers to control the vehicle's acceleration and deceleration. The most common ACC systems generally use automotive radar systems, placed at the front of the car, and/or a camera placed on the interior rear mirror. The radar is used to identify obstacles and predict their speed by sending and receiving radio waves. Camera-only ACC systems are currently being researched but are not yet state of the art. The ACC increases and reduces the car speed and automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead. The system may not react to parked, stopped, or slow-moving vehicles; it alerts the driver of an imminent crash and may apply limited braking but the main responsibility for steering the car lies with the driver.

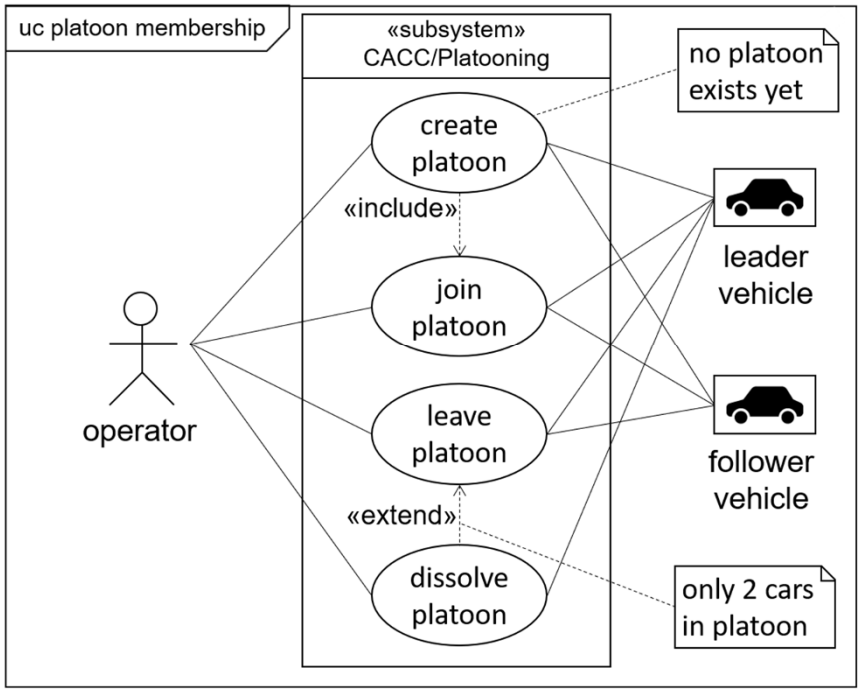


Fig. 1-1: *SysML use case diagram "Platoon Membership"*

Collaborative adaptive cruise control (CACC) takes the ACC technology to the next level, enabling vehicles to adjust their speed to the preceding vehicle in their lane with direct car-to-car communication. In the following, we use "CACC" to denote the cyber-physical system of communicating controllers in collaborating vehicles (that is, the collaborative system group (CSG)) and "CACC ECU" to denote the electronic control unit(s) in an individual vehicle (that is, the collaborative embedded system (CES)). Compared to

classical ACC, a CACC can respond faster to speed changes by preceding vehicles and even vehicles beyond the line of sight. These advancements improve the stability of the traffic flow, increase driver confidence, and allow distances to be minimized for vehicle-following. Ultimately, this results in better use of a highway's capacity and greater fuel efficiency. To increase efficiency by leveraging the collaborative aspect, the CACC may be observing several of the following common goals and targets:

- Same destination (at least partially)
- Support when driving on an unknown road/to an unknown destination
- Desired and steady cruising speed
- Reduced time and fuel consumption

Figure 1-1 shows the main SysML use case diagram[1] for platooning. Most of the collaborative aspects of the CACC functionality occur when a platoon is formed. Before any automated vehicle control can start, the vehicles have to notice each other and agree on a common driving strategy. During this phase, several aspects have to be considered:

- The vehicles must be in a close range so that a platoon can be formed physically. Therefore, the CACC must be aware of the physical location, speed, and direction of each vehicle. As a minimum, the vehicles must be aware of other CACC-capable vehicles and cars in their immediate vicinity.
- The vehicles must have a common driving direction. In the simplest case, the CACC would know the complete routes that all participating vehicles are about to travel. However, due to privacy concerns, this may not be the case; only partial information may be available from some vehicles.
- The vehicles should have a common or at least similar driving characteristic or goal. A truck platoon that wants to drive as economically and safely as possible might not be acceptable for a driver of a powerful car who wants to travel as fast as possible. Other drivers might not be willing to accept a very close distance to the surrounding vehicles, which is necessary to maximize the fuel savings. Such driving characteristics have to be negotiated between the participants.

[1] The term "SysML use case" should not be confused with the three use cases for collaborative embedded systems presented in this chapter. A SysML use case describes a dedicated functionality for a certain actor.

www.dbooks.org

- The vehicles must agree on their roles in a platoon. A lead vehicle (LV) has to be selected; all other platoon members will be assigned the role of a follower vehicle (FV). Either role might not be acceptable for some drivers. During the negotiations, a car can be a potential lead vehicle (PLV) or a potential follower vehicle (PFV).

A typical scenario for this use case is as follows. A vehicle drives on the highway and wants to create a platoon. The CACC ECU of this vehicle generates a platoon proposal and continuously broadcasts it to other vehicles that might join the platoon. Another vehicle's CACC ECU receives the proposal and accepts it. After the acceptance, both vehicles start a "platoon verification" routine, which includes a platoon role allocation (PLV and PFV). During the verification, no other vehicle can connect to the platoon. The PFV joins the PLV longitudinally at the rear. The speed of both vehicles is synchronized to establish a pairing. When the verification is closed and the platoon is created, PLV becomes LV and PFV becomes FV_1.

In the meantime, the platoon proposal remains active. Invitations for other cars to join the platoon are continuously broadcast. If a PFV_2 receives this request and accepts the proposal, the existing platoon will be extended by another FV. In the simplest scenario, PFV_2 must join at the rear of the platoon — in other words, behind FV_1. More complex scenarios would allow a vehicle to also join somewhere in the middle of an existing platoon. Assuming that the communication is organized as a peer-to-peer network, PFV_2 can pair with FV_1 or LV, depending on the platoon network topology. Once the pairing is finished, the platoon join is closed; PFV_2 becomes FV_2 and the platoon regulation takes control.

There are many more aspects and parameters that have to be considered or negotiated during the build-up phase of a platoon. As the vehicle platooning use case is considered in various chapters throughout the book, we do not go into detail here. Moreover, there are operations that may be reasonable but are not considered in this book, such as changing the order or the leader of a platoon, fusing two platoons into one, or splitting one platoon into two. A collaborative platoon management system has to be flexible enough to cope with such diverse information.

The CACC use case exhibits many challenges for advanced software engineering, described as typical for the development of CESs in Chapter 3: the complex functionality is realized mainly by software, there is a high degree of networking of heterogeneous components, and the system must act reliably and autonomously. Furthermore, the development must take into account common and

conflicting goals of the CESs. The challenges addressed in this book can be summarized as follows:

- Conception, implementation, and validation of a CACC that realizes the function of driving in a platoon
- Assessment of the quality of the platoon regulation concept, especially with respect to safety and reliability
- Platoon communication concept and its quality, especially the security
- Heterogeneity of CESs built by different vendors (and the resulting challenges for information exchange between these systems, including standardization)
- Means to cope with uncertainties caused by imprecise and possibly differing context perceptions of vehicles

Further challenges, which are not addressed here, include:

- Reliability of artificial intelligence techniques used for context perception and the related uncertainty
- Elicitation of requirements for engineering methods and tools for generalized collaborative car-to-car and car-to-X functionalities.

1.3 Adaptable and Flexible Factory



The use case “Adaptable and Flexible Factory” deals with production modules that collaborate to build products on demand. Each module consists of one or more production machines and offers one or more production functions (e.g., cutting, assembly, inspection, or forwarding of a workpiece). These functions can be combined in different ways, and even dynamically recombined according to changing customer needs. The common goal is to optimize the use of production resources and machines for different usage scenarios.

According to the VDI 5201 standard, flexibility and adaptability are concepts that describe *“the ability of manufacturing companies to change in response to changing general conditions. […] Adaptability refers to the ability to change involving structural changes to the system, while flexibility refers to the ability to change without structural changes.”* Present day industrial production facilities mostly consist of specialized production machines that are connected in a fixed way via stationary transport devices such as belt or chain conveyors. The need for adaptable and flexible factories is driven by several demands:

- Individualization and customization of products
- Variability of products in globalized markets

www.dbooks.org

- New or changed customer requirements
- Shorter product life cycles
- Changing markets and varying sales figures

Clearly, these demands cannot be met with traditional production systems. Adaptable and flexible factories are at the center of the fourth industrial revolution, comparable to the transition from individual manual production methods to mass production by machines in the 19th century. The ultimate vision of Industry 4.0 is to allow fully automatic production of individualized goods, reducing changeover times to zero. In order to realize this vision, several fundamental properties of a production system are required. The production process must be modular and arranged in several stages. Each production module must have a clearly defined set of capabilities and must be decoupled from other modules. Finally, the mapping of the process to modules and the topological layout of the process in the factory must be flexible. As most modern production facilities satisfy these requirements to some degree, the major obstacle to adaptable and flexible factories lies in the complexity of the corresponding systems engineering process.

Within this use case, we assume a factory is composed of multiple independent units called *production modules*. A production module can be thought of as a specific machine or device, or a tightly coupled group of machines. This covers both process industries and discrete manufacturing, where production modules are sometimes called *production cells*. Modules may be aggregated into different *production lines* that are substructures of a *production facility*. A *factory* may host several such facilities.

In our terminology, a production module or cell is a CES. The CSG is formed (statically or dynamically) according to a specific production job: it consists of all modules in the factory which take part in this particular production process. For a specific product component (e.g., a motor), this can be the corresponding production line. For a complete product (e.g., a car), the CSG consist of all modules in the corresponding production facility.

A production module is characterized by its ability to interact with the environment, which also includes communication with other production modules, humans (e.g., operators or maintenance engineers), and other entities within a factory (e.g., control systems or manufacturing execution systems). Collaboration arises from this possibility of interaction: several modules can form a production chain for a certain type of product. General functions of a module are:

- Processing
- Assembly
- Quality control — for example, visual inspection
- Transportation
- Storage of products

Flexible production modules are capable of performing different functions in the production chain. One example is a robot arm that can change the tool fitted (e.g., a welding gun) for another one (e.g., a digital camera). Adaptable production facilities are capable of changing the way the different modules are interconnected. An example is a mobile robot that can work in different production lines. This example shows that in an adaptable production facility, membership of a CES in a CSG can change dynamically.

In our use case, we consider a CSG for the production of quadrocopters. Each product consists essentially of components from five different classes:

- Mechanical sub-components
- Onboard electronic components
- Motors for the rotors
- Batteries
- Remote control units

Each of these components is available in several different variants, hence there are a large number of different products that can be built. The production process consists of several steps, which are performed either in sequence, in parallel, or independently of each other. Typical production steps are:

- Pre-assembly of rotor arms and rotor
- Pre-assembly of the body, including mounting of onboard electronics and battery
- Attachment of four arms and rotors to the body
- Final assembly of the full product

For each individual production step, activities such as turning, sticking, molding, drilling, screwing, etc. are necessary. The order of assembly of the different parts, and a production system which can realize this production task are shown in Figures 1-2 and 1-3.

www.dbooks.org

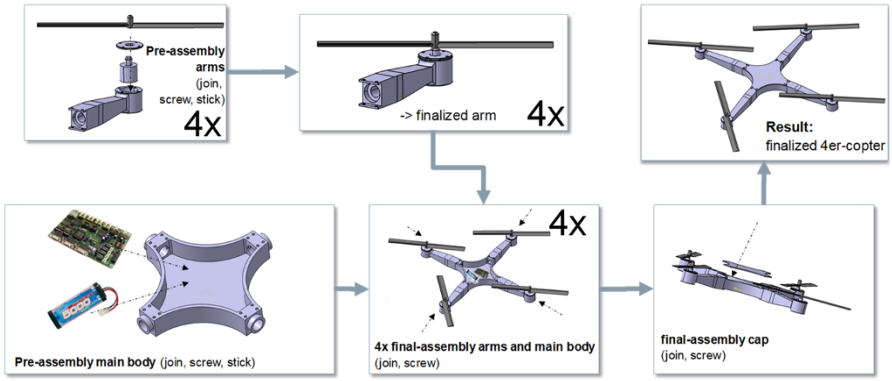


Fig. 1-2: *Process sequences "Quadrocopter" – order of assembly*

The production facility (i.e., the CSG) is structured into two main lines and several sidelines. Each line contains several production modules (i.e., the CESs). Each module is capable of performing different processing tasks (joining, sticking, gluing, soldering, etc.), allowing a flexible production of parts for different quadrocopters within one line. Moreover, the connection between sidelines and main lines can be adapted dynamically according to changing demands. Given a certain sequence of quadrocopters to be produced, the modules collaborate to accomplish this job as quickly as possible and with the most effective use of resources. Usually, this collaboration is orchestrated by a central manufacturing execution system (MES). The MES assigns each specific step of the production process to an individual production module and adapts the flow between the production lines accordingly. However, such a centralized control component is not really necessary; it would be feasible to imagine the production modules distributing the workload among themselves.

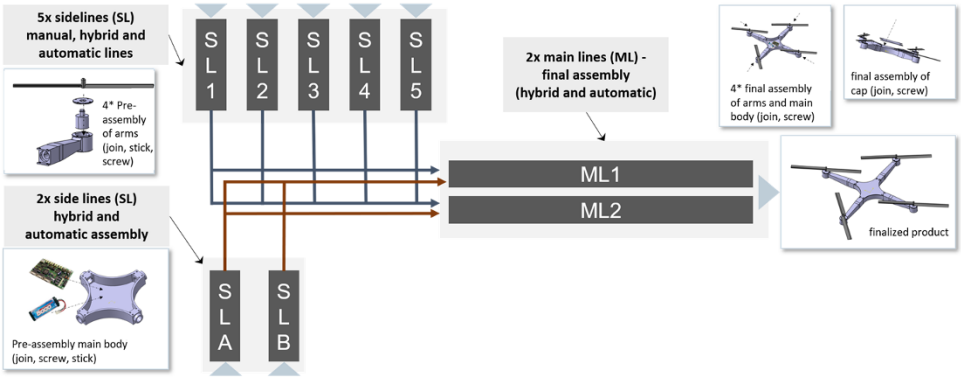


Fig. 1-3*: Example production system for the assembly of a quadrocopter*

The diagram in Figure 1-3 is an abstract model of the production facility. Given appropriate models of the production modules and their interconnections in the production facility, plus a description of the necessary production steps for each product and the estimated demand for each product, the best possible system configuration can be determined via simulation. In particular, simulation can be used to show the manufacturability of certain products or sequences of products, to determine the best timing of the modules and lines, to avoid bottlenecks and optimize the layout and output of the facility, and to calculate the cost per unit and management costs. Chapter 12 shows how to create adequate models for this use case.

Challenges for the design of adaptable and flexible factories, which are addressed in this book, are as follows:

- Definition of engineering methods and a corresponding process for the design of an adaptable and flexible factory
- Integration of qualities into the engineering methods and models — for example, safety, reliability, and security
- Creation of models for production modules and facilities
- Description of production processes and validation of orders
- Simulation and analysis methods for these models:
  - For proving properties of the CESs as well as the CSG
  - For managing variability in the CSG
  - For risk assessment and risk decomposition
- Engineering tools that support the adapted engineering methods
- Migration concept for converting a legacy production site into an adaptable and flexible factory step by step



1.4 Autonomous Transport Robots

Our third use case deals with autonomous transport robots, which are driverless vehicles for loading and unloading production modules in a factory. Since they are not stationary, autonomous transport robots can realize the material flow between flexible units in an adaptable production facility. In our terminology, each robot is a CES, and the fleet of robots is the CSG that provides the transport service to the production facility. We explore a decentralized control scenario, where each robot can decide which transport job to accept and accomplish. The common goal of the fleet is to keep production going — that is, no production module may ever stop due to lack of supply material or abundance of processed material.

www.dbooks.org

In present-day factories, traditional transport systems such as conveyor belts or rollers are increasingly being replaced by automated guided vehicles (AGV). The task of these AGVs is to provide an automated flow of material between storage, machinery, workspaces, and shipping department — for example, to transport small load carriers, trays, barrels, and coils. Moreover, they can be used for the automated transport of components to quality control or refinishing operation spaces, and for the transport of tools and testing equipment to assembly lines or working spaces.

The advantages of AGVs in comparison to stationary conveyor systems are:

- Scalability: A fleet may grow as necessary with regard to the number of transportation tasks. If business demands grow, new vehicles can be added to the fleet easily.
- Changeability: The layout of a production process can be changed easily, as no stationary equipment has to be rebuilt.
- Fault tolerance: With stationary equipment, even a small failure of a single part often means that the whole process is halted. If one of several AGVs malfunctions, however, the others can simply take over its tasks.
- Reduced space: In general, vehicles use less space than conveyors; moreover, they can be stowed away if not in use. In fact, as modern transport robots use the same walkways as human factory workers, the additional space requirements are minimal.
- Easy deployment: Since there is no construction work necessary, AGVs can be deployed at a production site within a relatively short amount of time.

The first generation of AGVs, introduced in the 1950s, were capable of following a white line or other optical markers on the floor. They used to drive on circular one-way routes on dedicated lanes in the factory. Thus, there were only a few advantages compared to stationary conveyor systems. The second generation, which emerged around 1970, still had to use dedicated areas that humans were not allowed to enter but could localize themselves in these areas via photoelectric and inductive sensors. Thus, they could move more or less freely within a blocked segment of the traffic route, which allowed more flexibility. Laser scanners for distance measurement became available in the 1990s, with safety features available only from the 2000s. A rotating laser scanner for distance measurement can not only stop the AGV if a person approaches, it can also build a digital map of the factory environment and allow the AGV to move freely in the facility.

An autonomous transport robot is an AGV that can navigate autonomously. It does not require any kind of markings, reflectors, or track guidance. Using a pre-recorded map of the environment, it finds its path by itself, without the need for fixed routes on traffic ways. Localization is done via comparison of the data from the integrated laser scanner with an internal map of the factory. Routing is also autonomous: when a robot receives an order to transport a load from point A to point B, it uses the map to calculate an optimal path. In the case of there being an unexpected obstacle on this path—for example, a pallet that the vehicle cannot circumvent—the robot comes up with an alternative route. If no alternative exists, the robot reports to the central management software that the order cannot be executed.

In this use case, we consider a fleet of autonomous transport robots as a CSG. Currently, transport robot fleets are managed and controlled centrally. A fleet organization system AIC (AGV interface controller) is in contact with the customer's manufacturing execution system and translates material requisitions into transportation tasks for the fleet. Criteria for the AIC's choice can include the vehicle's distance to the pick-up-area, avoiding robots driving without a task, and the battery status of the robots. From the AIC, the robots receive simple instructions with a "pick up here, carry there" structure and then plan the route to get to their destination, with each robot taking little individual action and robots gathering information first and foremost from the central controlling system.

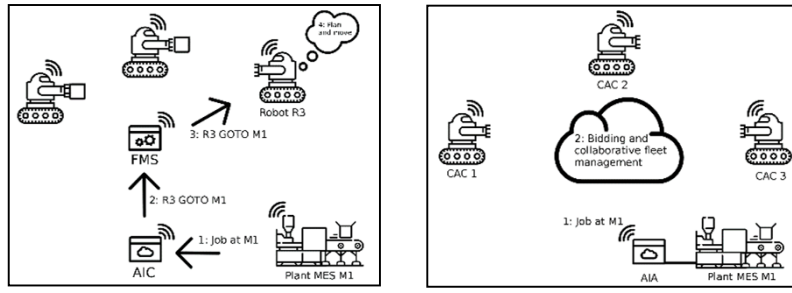


Fig. 1-4: *Central and decentralized fleet management*

Here, we are considering transport robots as individuals with goals, foresight, and an awareness of the other robots in the fleet. Individual robots are granted a higher level of autonomy, and the central AIC is no longer necessary. The task management system merely offers tasks that must be performed, and the robots distribute these amongst themselves according to individual capabilities (see Figure 1-4). This has several advantages. Among other improvements,

www.dbooks.org

it increases the overall efficiency, making more sensible use of resources and moving in ways that ensure no robot becomes a hindrance for others.

The user story in Table 2-5 describes how autonomous cooperating robots can determine which one of them fulfils an order for transportation. If a new order is given and several robots are available to take it, there must be a decision about which one of these robots will actually perform the task. This can be accomplished via a "bidding" process in which each robot calculates its factors playing into this task — for example, how far away it currently is from the pick-up area or what its current battery charge status is. It then sends these combined factors to the other robots as a bid. Depending on which robots can offer the most practical circumstances, a distributed consensus protocol is used to decide which robot takes the order.

Table 2-5: *User story for distribution of transport jobs*

	Who?	**What?**	**When?**	**Why?**
1	Production Module	Broadcasts transportation need to robots	Every time a module has need of support or availability (may be in advance and/or may be with priority)	The production process of the module is not allowed to stop
2	Every Robot	Calculates a bid for this transport (may be based on individual cost and/or other criteria)	When a new transport need is notified	To get the information about which robot is the best fit for this transport
3	Every Robot	Determine winner by distributed leader election algorithm	After bidding	
4	Winning Robot	Adds the transport to its own transport queue	When a bid is won	That the transport need is satisfied

Further challenges in this use case are as follows:

- Cooperative path planning: Ideally, each robot should share the information about blocked paths with the other robots in the fleet. This information must be updated at frequent intervals. A more advanced option would allow path planning according to the traffic situation and the presumed paths of the other robots.
- Fault tolerance: The transportation system is not allowed to halt if some of the robots are offline (in a dead spot where there is no wireless reception) or cannot localize themselves because of massive differences between the observed and expected environment.
- Flexible fleet size: It should be possible to integrate a new robot into an existing and operating fleet without stopping production. After it has authorized itself, the new robot receives map and task information from the others and is able to collaborate in the fleet as a coequal partner.
- Distributed logging and monitoring: For a possible "transport as a service" operation mode, the fleet must remember all relevant transactions. The logging of this data must be safe and secure—for example, via a block chain mechanism.

Acknowledgement: The author wishes to acknowledge the contributions to the CrESt use case descriptions by the authors of the respective deliverables, in particular Oliver Kreuzmann, Stefan Penz, Jorge Castillo, Suryo Buono, Birthe Böhm, Roland Rosen, Jan Vollmar, Jan Christoph Wehrstedt, Wolfram Klein, Sebastian Schröck, Constantin Hildebrandt, Alexander Ludewig, Birte Caesar, Marian Vorderer, Michael Hassel, Sebastian Törsleff, Jan Winhuysen, Tobias Schüle, Michael Nieke, Jan Stefan Zernickel, Susanne Dannat, André Schmiljun, Henry Stubert, and Janina Samuel. Moreover, the author thanks the reviewers Torsten Bandyszak, Birthe Böhm, Birte Caesar, Alexander Hayward, Jörg Kirchhof, Vincent Malik, Nikolaus Regnat, Sebastian Schroeck, and Jan Christoph Wehrstedt for their valuable remarks.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Birthe Böhm, Siemens AG
Wolfgang Böhm, Technical University of Munich
Marian Daun, University of Duisburg-Essen
Alexander Hayward, Helmut-Schmidt-Universität
Sieglinde Kranz, Siemens AG
Nikolaus Regnat, Siemens AG
Sebastian Schröck, Robert Bosch GmbH
Ingo Stierand, OFFIS e.V.
Andreas Vogelsang, Technische Universität Berlin
Jan Vollmar, Siemens AG
Sebastian Voss, fortiss GmbH
Thorsten Weyer, University of Duisburg-Essen
Andreas Wortmann, RWTH Aachen University

2

Engineering of Collaborative Embedded Systems

Embedded systems are being increasingly used in changing environments where they no longer fulfill their associated stakeholder goals on their own, but rather in interaction with other embedded systems. This transition to networked, collaborative embedded systems is creating new application opportunities that impose numerous challenges for developers of these systems. In this introductory chapter of the book, we present the complexity of these systems and the challenges associated with them in a coherent manner. We illustrate the challenges using two use cases, "Vehicle Platooning" and "Adaptable and Flexible Factory." Finally, we reference the challenges of developing collaborative embedded systems to the individual chapters of this book, which describe various methods of mastering the complexity in more detail.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_2

2.1 Introduction

New class of systems

With the transition from classical embedded systems to networked, collaborative embedded systems (CESs), new applications for industry are emerging. The ability of a company to efficiently develop CESs of the highest quality will therefore become a decisive competitive factor. At the same time, this transition will lead to a leap in the complexity of the systems under consideration. Not only single embedded systems, but also dynamically changing networks of CESs at runtime have to be considered. Since the success of products in the area of embedded systems is strongly determined by their quality, it must be possible to guarantee a high system quality despite the increasing complexity. Therefore, it is essential to be able to control the complexity of CESs with efficient methods. This includes suitable methods for specification, implementation, and validation of these systems. The development of CESs goes hand in hand with important safety and security issues, which have to be addressed comprehensively for a broad industrial application by relevant development approaches.

This chapter gives an informal introduction to the challenges of developing CESs. We start with the definition of important terms and then describe the challenges that have to be overcome in the development of such systems. These challenges are explained in more detail by means of two use cases. Finally, at a high level of abstraction, we provide an overview of selected results achieved in the CrESt project[1]. Details of the results can be found in the corresponding contributions of this book (see Section 2.6 and the Appendix).

2.2 Background

Model-based systems engineering

Model-based systems engineering (MBSE) [Selic 2003] aims to reduce the conceptual gap between problem domains (mechanical engineering, automation, biology, law, etc.) and the solution in software [France and Rumpe 2007], and to integrate contributions from the participating domains. For this purpose, models—often in the terminology of problem domains—are used as documentation. Furthermore, development artifacts that reduce this gap with an explicit description of problem domain concepts can be accessed with

[1] Website of the CrESt project: https://crest.in.tum.de/ (available in German only)

www.dbooks.org

sufficient formalization of efficient automation. These artifacts also simplify the integration of contributions from different domain experts by abstracting solution domain details.

In the German Federal Ministry of Education and Research (BMBF) project SPES2020 [Pohl et al. 2012] and the follow-up project SPES_XT [Pohl et al. 2016], significant results for MBSE have been achieved that further advance the development of highly automated embedded systems and have already established themselves as a methodological approach in industry.

The two SPES projects provide a methodological toolkit for MBSE that allows an efficient model-based development of embedded systems. At the same time, the toolkit is based on a solid scientific foundation with a special focus on consistency and semantic coherence (see [Broy and Stolen 2001], [Broy 2010]). The SPES methodology building set is based on three principles of outstanding importance:

Principles of the SPES methodology

- ❑ Consistent consideration of interfaces along the design process
- ❑ Decomposition of the interface behavior and the description of systems via subsystems and components at different levels of granularity
- ❑ Definition of models based on the previous points for a variety of cross-sectional topics (variability, safety, etc.) and analysis options

In SPES, a system model is a conceptual ("generic") model for describing systems and their properties. It describes what constitutes a system as the result of a conceptualization. System models define the components of the system and its structure, the essential properties, and other aspects that have to be considered during development. Among other things, system models define what requirements refer to (subject of discourse). In SPES, the system model consists of (see Figure 2-1):

System model

- ❑ An *operational context*[2] that influences or is influenced by the system at runtime
- ❑ An *interface* that clearly separates the system from its operational context
- ❑ A *behavior* of the system that can be observed at the interface (indicated by arrows at the interface)

[2] SPES distinguishes between the operational context, which in turn consists of the structural, functional, and behavioral context, and the knowledge context (see e.g., [Pohl et al. 2016]). In this chapter, however, only the operational context is relevant.

- An *inner structure* of interrelated and communicating elements (architecture), which are themselves systems

The system model used in SPES is static in the sense that model elements do not change at runtime. This applies in particular to the appearance and disappearance of elements in the operational context as well as the adaptation of the system interface at runtime. With the transition from classical embedded systems as considered in SPES2020 and SPES_XT to networked CESs, new applications for the MBSE approach arise. At the same time, this transition leads to a leap in the complexity of the systems under consideration. Not only single embedded systems but also networks of CESs have to be considered. Such system networks can be constituted dynamically at runtime by a multitude of different embedded systems (homogeneous or heterogeneous type) [Grosz 1996]. In their interaction, these system networks enable the users to achieve a comprehensive added value that goes beyond the benefits of the individual systems. In such systems, both the exact system configuration (i.e., the system boundary) and the system context at design time can only be anticipated with considerable uncertainty. In the context of the development of CESs, this raises new and important questions

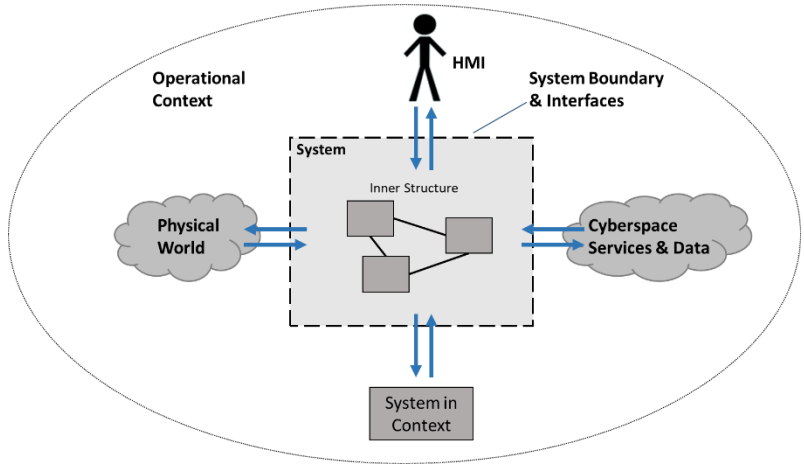


***Fig. 2-1**: SPES system model*

regarding the functional safety of the systems and the dynamically formed system networks [Damm and Vincentelli 2015], [SafeTRANS 2019].

www.dbooks.org

The BMBF project CrESt[3], which was conceived as a continuation of the work of SPES2020 and SPES_XT, addressed these new challenges and the increasing complexity in the development of highly automated embedded systems and developed the SPES framework for MBSE with regard to CESs.

2.3 Collaborating Embedded Systems

2.3.1 Collaborative and Collaborating Systems

With the term collaboration, we denote the (active) interaction of several embedded systems in one system network. The purpose of collaboration is to achieve a common goal through the mutual provision of functions that individual systems alone cannot achieve [Broy and Schmidt 2001], [Sha et al. 2008]. Collaboration therefore serves to achieve the goals defined in a single system or a system group and can take various forms with regard to possible binding times, the type of coupling, the process of forming the group, or the collaboration management. From the point of view of our system model, it is not so easy to distinguish collaboration from "simple" interaction. In fact, collaboration must of course manifest itself at the interfaces of the collaborating systems in the form of interaction.

A collaborative system can therefore be distinguished from a non-collaborative system not so much by the system model as by its origin, its use, and its purpose. Maier has defined two properties that must apply to collaborative systems (as opposed to non-collaborative systems) [Maier 1998]:

- Operational independence of elements: The systems involved in a collaboration provide added value even if they are operated independently of the collaboration.
- Managerial independence of elements: The systems involved in a collaboration are actually developed and operated independently.

Collaborative embedded system (CES)

Taking these properties into account, we define a *collaborative embedded system* (CES): CESs are embedded systems that can collaborate with other CESs to achieve goals that are difficult or impossible for a single CES to achieve alone.

[3] Funded by the German Federal Ministry of Education and Research under the funding code 01IS16043

Collaborative system group (CSG)

A *collaborative system group* (CSG) is formed dynamically at runtime by a set of CESs that collaborate with each other. The CESs involved can take on different roles in the group. It is important to note that a CSG can also be seen as a system in the sense of Figure 2-1, where the internal structure is formed by the collaborating CESs.

Collaborating systems

We call a CES a collaborating CES if it is actively involved in a CSG at a certain point in time. Note that a system can be collaborative for a certain CSG type (e.g., platoon), while it is not collaborative for another CSG type (e.g., adaptable and flexible factory).

Note that a CSG and the CESs are at different levels of granularity

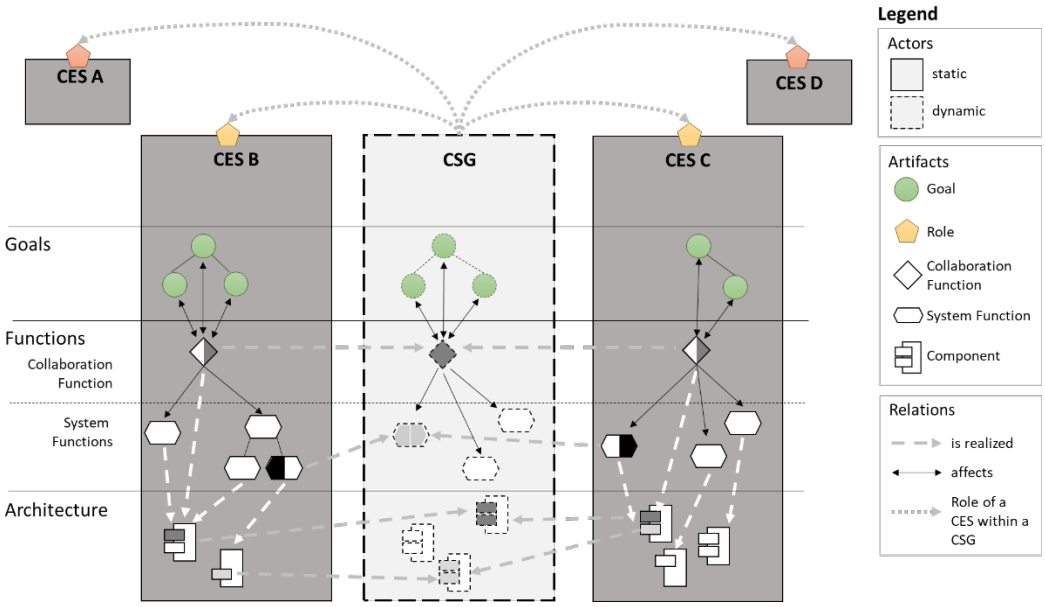


Fig. 2-2: *Goals, functions and architectures in collaborative system groups*

in the SPES modeling framework (see [Pohl et al. 2016]): while the CSG models describe the overall system and are thus located at the highest level of granularity, the CES models are located at the next level of granularity of the framework, and thus represent architectural components (subsystems) of the CSG.

www.dbooks.org

CESs can be developed and realized with the help of methods defined in CrESt[4]. The most important concepts for the collaboration of CESs are illustrated in Figure 2-2.

2.3.2 Goals of System Networks

CES goals

In addition to the CESs, the CSGs also have goals that are negotiated when the CSG is formed. This involves checking whether there is sufficient agreement with regard to the achievement of the goals of the participating CESs. We differentiate between negotiable goals ("soft goals"), which can be adjusted if necessary to allow the CES to participate in a CSG, and non-negotiable goals ("hard goals"), which, if they conflict with the goals pursued by a CSG, may result in a CES being unable to join a CSG. Goals can also be refined hierarchically. Furthermore, relationships can be used to define dependencies between goals. The set of goals pursued by a CES, as well as the relationships between the individual goals, form the goal system of a CES, which is already fundamentally (generically) defined during development. This goal system is then individually instantiated at runtime in the respective CES instances, thus concretizing the goals.

CSG goals

During the conceptual development of a CSG, a basic goal system consisting of soft and hard goals is also defined. This goal system contains overarching goals that can only be achieved within the CSG through cooperation between the CESs involved. At runtime, the CSG goal model is instantiated by goal negotiations between the participating CESs: the overarching goals are specified and compared with the individual goals of the participating CESs. Within the collaboration, the participating CESs make their system functions available to each other in order to achieve common goals that they cannot achieve on their own. If conflicts arise—for example, between the overarching goals of the CSG and the individual goals of the participating CESs—these must be resolved.

2.3.3 Coordination in System Networks

Control mechanisms

Where different CESs contribute collaboratively to a CSG goal, it must be ensured that the individual contributions are coordinated and aligned. Different control mechanisms are conceivable here. For

[4] For a better distinction between CES and CSG, we assume in the following that CESs, unlike CSGs, are "static" in the sense that their functional scope and physical architecture are already fully known at the time of design. In particular, this excludes the possibility of nesting system networks of CSGs.

example, the collaboration of several systems within a CSG can be centrally controlled by the role of a coordinator. The CSG coordinator can also decide when and under which conditions other CESs join or leave the CSG. In contrast, collaboration within a CSG can also be organized decentrally. Depending on how critical the contribution of individual CESs is to the common goal, their commitment to the CSG will also be more or less firm: for example, a CSG can forbid its participants to leave the group before certain common goals have been achieved.

Whether a CSG is to be managed and organized centrally or decentrally depends on the circumstances of the respective domain on the one hand, and on the other hand, on the roles the CESs can take within the collaboration. In special cases, it may be necessary to prepare CESs for collaboration through structural design. Should these CESs wish to enter a new CSG in order to collaboratively contribute in another way to a new collaborative goal, a reconfiguration might be necessary that can only be performed by an external actor and for which the CES or even the CSG has to be temporarily taken out of service.

2.3.4 Dynamics in System Networks

In the following, dynamics is understood to mean both the dynamics within CSGs and the dynamics within their operational contexts. Dynamics refers to the change over time in structure and behavior over time.

Dynamicity of the CSG

The concrete inner CSG structure is dynamic, since new CESs can join and leave the CSG at runtime. For example, at design time, there is no definition of how many CESs are currently involved in the CSG. At this point in time, only the types and roles of the participating systems and the basic structure of the CSG are defined.

Dynamicity of the operational context

The operational context is also dynamic, since systems can enter and leave the context. The special consideration of dynamics here is that there is transition between system components of CSGs (CESs) and their context. This includes, for example, the fact that individual systems in the context of a system join together to form a CSG. The question as to which objects belong to the context of a system and which do not (i.e., which CESs are part of the CSG) depends on their relevance in connection with the fulfillment of the CSG goals.

www.dbooks.org

Open world assumption

For dynamic systems, depending on the application domain, openness in the sense of the *open world assumption*[5] plays an important role [Keet et al. 2013]. In contrast to the *closed world assumption*, there is no assumption here that the possible states of the operational context are completely known from the outset, but rather that states may exist that are unknown to the system. This view has important consequences. For example, an object recognition algorithm would not be able to reject an unknown object as a possible malfunction but would have to recognize "unknown" objects. In other words, openness is understood to mean the property that the environment in which a CES or CSG is to operate is not fully known at design time. If, from the perspective of the CES or CSG, the context in which it is expected to operate is not fully known, this is called openness of context. If the structure of the CSG itself is not fully known at the time of development, this is called openness of CSG. Openness can refer to object instances and object types. The former allows the occurrence of additional objects of previously known types, while the latter also allows the occurrence of objects of new types.

Uncertainty

As a further consequence, CESs and CSGs formed at runtime that operate in open contexts must be able to deal with imprecise, contradictory, uninterpretable, and even missing context information [Bandyszak et al. 2020]. The phenomenon of such "fuzzy" information about properties of the CES or CSG context is characterized by the term "uncertainty" of context information. CESs and CSGs must be able, whenever necessary and possible, to mitigate the existing uncertainty individually or within a CSG — that is, to dissolve the uncertainty completely, reduce it, or take appropriate measures to continue to operate reliably and robustly in the context of the given uncertainty.

Conceptual definition

At the design stage, CSGs are developed conceptually — for example, in the form of standardization of interfaces and the definition of basic architectural decisions and concepts for the formation of a CSG. This defines the type of the CSG and its abstract properties and goals. However, the overall system behavior and the complete architecture of a CSG can only be specified after instantiation and only taking into account the CESs. This concrete realization by CESs only takes place dynamically at runtime through the interaction of the collaborating CESs involved. All prerequisites

[5] The "closed world assumption" describes the principle that only events that were considered at design time occur in a context and that other events should be treated as failures.

that a CES must fulfill in order to participate in a CSG must be described conceptually at development time. If a CES assumes one or more roles in a CSG, it provides the system group with the necessary functions.

The formation of a CSG must be designed and specified conceptually during the development phase — both at the level of the CESs and at the level of the CSG at various levels of detail by describing the necessary interfaces and protocols. This ensures that the CESs have a common definition of the communication (suitable protocols and interfaces), of roles to be assumed and their interaction in the CSG, system functions to be provided, and other quality requirements of the CSG during runtime. Here, too, the respective domains specify the level of detail to which a CSG is planned during this concept development and the extent to which knowledge about the nature of the CSG differs between the CESs (potentially involved).

2.3.5 Functions

Function interface

In order to fulfil the goals defined in the CESs and CSGs, different functions that must be implemented in the CESs are required. A function can be described at its interfaces by inducing a certain behavior on the basis of predefined, possible inputs and thereby generating different outputs [Broy and Stolen 2001]. The current implementation is encapsulated by the interface and the input/output behavior. For functions to actually be executed, they must be implemented in an architecture.

System functions

We distinguish (logically) between two classes of functions: one subset is formed by the system functions, which can be represented at very different levels of detail and represent the concrete end-to-end added value compared to the system context and to the achievement of the CSG or CES goals that a CSG or CES is capable of providing.

Collaboration functions

The second class consists of the collaboration functions: prior to collaboration, the CESs must communicate with each other, exchange information about their possible contributions in the form of system functions, communicate and, if necessary, adapt, negotiate, and prioritize their goals, and define the concrete course of the collaboration. This requires a comparison between the requirements for goal fulfillment and available system functions. The role that each CES will take on within the CSG must also be determined before the actual collaboration takes place. This depends, among other things, on which role a CES can generally take on due to its functional nature. All these basic functions for the realization of a collaboration, and

www.dbooks.org

especially the alignment between goals and system functions, are summarized in the model as collaboration functions.

A collaboration function differs from the system functions in that it does not so much represent the individual contribution of the CES, but rather provides the functional basis for enabling the collaboration. Every CES must have collaboration functions in order to be able to collaborate in principle, regardless of which concrete system functions it contributes to a collaboration. Which CESs within a CSG communicate with each other and which hierarchies exist to make decisions depends on whether the CSG is organized centrally, with fixed hierarchies, or decentrally.

Emergent behavior

In either case, the CSG is a construct that is pre-designed at design time, implemented in the CES, and dynamically assembled at runtime. Both the goals of a CSG and its functions are aggregated components that are implemented only in the CES. Thus, a CSG function for achieving a CSG goal consists of a combination of system functions of several CESs involved in the collaboration or, if applicable, of one or more system functions of a CES. The coordinated execution of the system functions of the CESs generates the behavior that realizes the CSG function. This behavior can also be described as emergent, since the CSG functions may create new properties of the CSG as a result of the interaction of its collaborating elements. The emergent properties of the CSG cannot always be directly—or at least not always obviously—traced back to properties of the CES, which they have in isolation.

CSG architecture

Just as the functions of a CSG are aggregated from the functions of at least one CES, the CSG architecture is formed just as dynamically from the static architectures of the CESs. In contrast to the static architecture of a CES, which can be developed and planned explicitly, the dynamic architecture of a CSG is again planned conceptually and the necessary elements are provided in the architectures of the CESs. This allows a comparison as to whether a CSG includes the corresponding architectural elements that are necessary to achieve the overall goal of the CSG. Only in this way is it possible to dynamically form different CSGs at runtime.

2.4 Problem Dimensions of Collaborative Embedded Systems

With the consideration of collaborative embedded systems in CrESt, two further problem dimensions are added to the systems considered

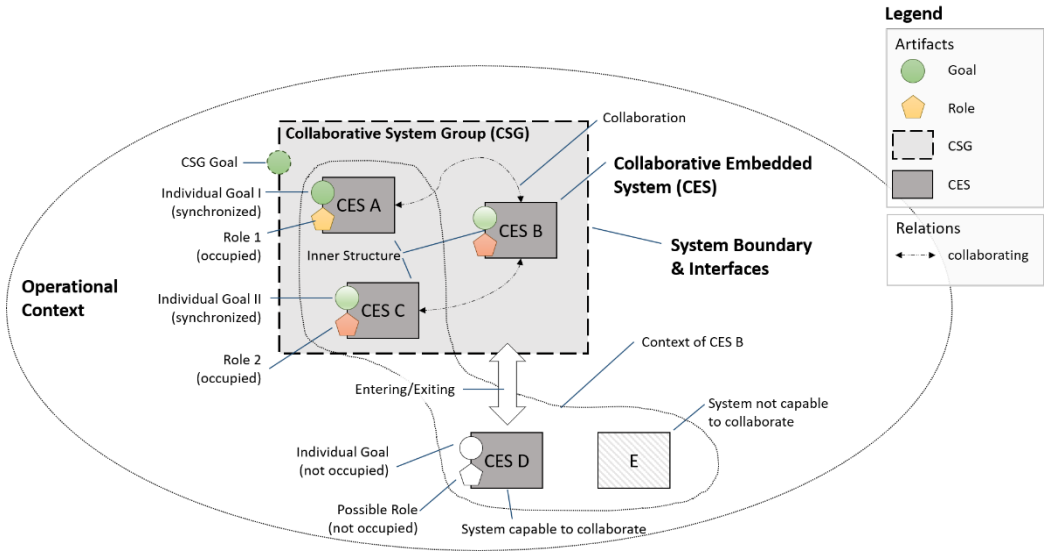


***Fig. 2-3**: Collaborative embedded systems at a glance*

in SPES: collaboration and dynamics at runtime. A distinction is also made here between CESs and CSGs, which form these embedded systems at runtime. In the following sections, we explain these additional dimensions in detail, especially with regard to the system concept. Figure 2-3 provides an overview of the most important terms and relationships.

In addition to the classic characteristics of the system model, such as interface, observable behavior at the interface, operational context, and internal structure (see Figure 2-1), communication between CESs, system goals, and the role of the CES in the system network must also be considered in the case of CESs. Furthermore, the characteristics are no longer statically and completely known at design time but can change at runtime. For example, when a CES enters the CSG, its internal structure changes. At the same time, the operational context of both the entering CES and the CSG changes. For systems in context, we also distinguish between collaborative systems—that is, systems that are able to enter a CSG due to their architecture and functions—

www.dbooks.org

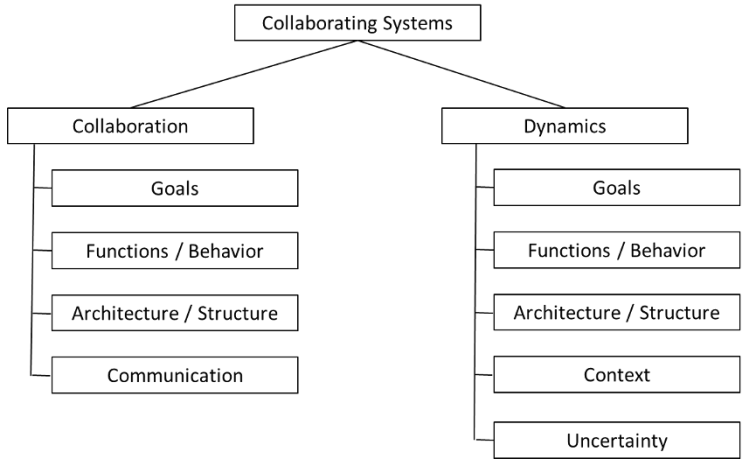


***Fig. 2-4**: Taxonomy of CrESt challenges*

and non-collaborative systems that cannot take an active role in the CSG at any time.

Based on these specific challenges for collaborative embedded systems, a taxonomy of challenges can be defined, as shown in Figure 2-4. For the two superordinate categories "*Collaboration*" and "*Dynamics*," a number of characteristics were defined. The challenges are described in detail in the following sections.

2.4.1 Challenges Related to Collaboration

CESs must be designed in such a way that they can operate in conceptually conceived types of CSG. This requires both the communication of objectives and the ability to take on different collaborative roles and act accordingly. To this end, these systems must be able to make their system functions available to CSG and - also in terms of quality - to communicate them to other CESs at runtime. For this purpose, collaboration is considered under the following aspects:

Goals

- *Goals*: A CES must be able to align its individual goals with the goals of CSG. In doing so, the CES must decide what contribution it can make to the common goals and which individual goals, if any, must be adjusted (see "Hard Goals" and "Soft Goals" in Chapter 2).

Functions / behavior

- *Functions / behavior*: A CES must be in a position to provide CSG with its own system functions. In addition, options must be

provided for how it can adapt its own functions and qualities within the framework of the negotiated CSG objectives.

Architecture / Structure

- *Architecture / Structure*: A CSG is an initially virtual entity that is thought of at design time and then forms (and can dissolve) dynamically at runtime. At design time only a conceptualization takes place. It is realized through the interaction of the participating CES and their architecture components.

Communication

- *Communication:* The basic ability of the CES to communicate with other CESs is realized by means of the collaboration functions. Among other things, these functions also form the basis for negotiating objectives, assigning roles and communicating available system functions to CSG.

2.4.2 Challenges Related to Dynamics

The developers of dynamic CESs need concepts and methods that support their design so that they can operate in a highly dynamic and possibly open operational context in dynamically formed CSGs at runtime and, if necessary, with "fuzzy" information in a targeted manner. As shown in Figure 2-3, the context of a CES differs significantly from that of a CSG according to its hierarchical structure. The context of a CSG consists exclusively of surrounding systems of the CSG, while the context of a CES is formed by the other CESs of a CSG and, if applicable, by parts of the context of the CSG.

As shown in Figure 2-2, the system and collaboration functions of CSG are formed by the functions of the CESs. A challenge to a CSG must therefore always be solved by the CESs involved, possibly also by the interaction of several CESs. Dynamics, openness and uncertainty give rise to the following challenges for the development of these systems, among others:

Goals

- CSGs must be able to adapt their goals to the changes in the operational context that they perceive via their sensor technology. This is particularly the case when CESs become part of CSG or leave CSG. This requires the possibility to dynamically adjust the goals of the CESs (see challenges on collaboration goals).

Functions / Behavior

- CSGs must be able to cope with changes in available functions. This concerns on the one hand the system functions of the CESs in an operational context and on the other hand the collaboration functions of the CESs within the CSG. For this purpose, CESs must be able to describe their available system or collaboration

www.dbooks.org

functions and to adapt their system functions according to the negotiated goals. For example, a CES must be able to communicate with new objects that have been added to its CSG or operational context. In open systems (in the sense of the open world assumption), this requires the ability to communicate with systems of previously unknown types and, where this is not possible, to handle them safely in other ways. At the CSG level, it must be possible to describe the functions required to achieve certain goals in the form of sought-after capabilities and to search for these in their CSG and operational context. They must be able to recognize when new system functions are available and, if a system function is no longer available, search for alternatives. Finally, a CSG must be able to select available alternative system functions according to defined criteria.

- ❑ The CESs of a CSG must be able to deal with structural changes of the CSG. In particular, they must be able to detect changes in their operational context and CSG and adapt their interfaces accordingly. The dynamic entry or exit of a CES from the group or the change of roles of the CES within a CSG also affects the internal structure of the CSG. *Architecture / Structure*
- ❑ Both CESs and CSGs must be able to deal with changes in the operational context of the CSG or of CESs within the CSG. To do so, they must be able to consider the behavior of context objects (of the CESs and CSGs) when planning their own goal fulfillment and implementing desired functions. To analyze the current context behavior and the potential changes resulting from it, it is necessary to define the desired or expected behavior of context objects. Finally, CESs must be able to evaluate behavioral changes in terms of their impact on the CES or CSG under consideration and to draw conclusions from this. This includes, for example, the adjustment of goals. *Context*
- ❑ Dealing with uncertainty and fuzziness in data collection is also relevant for classical systems. For dynamic systems, which should be able to operate in open contexts, it must be possible to resolve or deal with uncertainties in terms of the open world assumption. *Uncertainty*

2.5 Application in the Domains “Cooperative Vehicle Automation” and “Industry 4.0”

In the following, we consider and concretize the challenges on the basis of exemplary application domains.

2.5.1 Challenges in the Application Domain "Cooperative Vehicle Automation"

The use case "Cooperative Vehicle Automation" investigates system networks that are formed between vehicles in order to achieve common goals. An obvious scenario in this context is "vehicle platooning" (computer-controlled convoy driving). In the sense of CrESt, this is a system group (i.e., a CSG) of individual vehicles (CESs) that drive in close succession in close proximity to each other with the aid of automated control systems. Often, the common goal of such a network is to reduce the fuel consumption of all participants and to relieve the individual drivers. During their participation in a platoon, individual vehicles coordinate their own goals with the common goals. For example, individual vehicles with individual destinations for a certain route can join a platoon that has a different final destination but is travelling in the same direction.

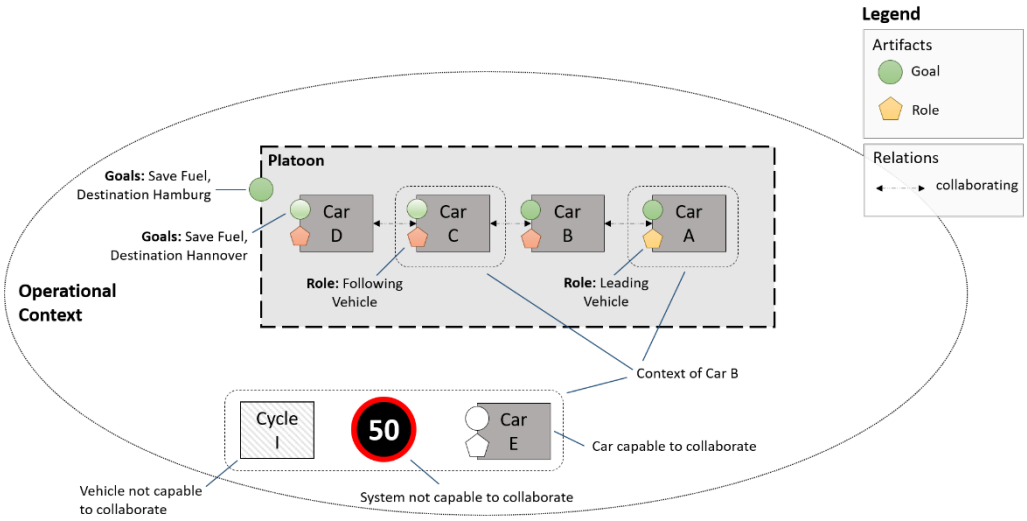


***Fig. 2-5**: Overview of collaboration in computer-controlled convoy driving*

Figure 2-5 shows an example of the structure of such a platoon, consisting of vehicles A to D. Car A, at the head of the convoy, takes on the central role of coordination, referred to here as the "leading vehicle." In this role, the vehicle coordinates basic tasks such as the creation and dissolution of the platoon, or processes such as the execution of a lane change for the entire platoon. The other vehicles take on the role of "following vehicle" and thereby transfer part of their control to the lead vehicle. In addition, individual vehicles of the

www.dbooks.org

platoon can also contribute further system functions. This allows new sub-functions of the platoon to be formed and the overall functionality of the platoon to be expanded. For example, a vehicle could bring special sensors into the platoon for better environmental monitoring, which are then available to the platoon as a whole.

In order for a platoon to be formed, certain requirements must be met by the participating vehicles. The preliminary design phase for platoons must therefore define which requirements, such as wireless communication connections, standardized communication protocol, suitable distance sensors, must be met by the vehicles of a platoon.

Collaboration

In the following, we look at the specific challenges in the area of collaboration using the example of a vehicle entering a platoon. Car E wants to enter a platoon consisting of four vehicles (see Figure 2-5).

Goals

Car E must coordinate its individual goals, such as destination and cruising speed, with the platoon's common goals before entering. The cruising speed is a soft goal, so Car E is allowed to adjust its speed to the cruising speed of the platoon.

Functions and behavior

Upon entry, Car E is assigned its future role (usually as a following vehicle) in the platoon. It must adapt its behavior to this role. For example, decisions on initiating acceleration, braking, and lane changing processes are transferred from Car E to the lead vehicle.

Architecture and structure

When entering the platoon, Car E will give an entry position. This specification can influence and optimize the structure of a platoon — for example, for an imminent exit of another vehicle.

Communication

For the entry of Car E into the platoon, extensive communication between Car E and the platoon's lead vehicle is necessary. Car E has to express its wish to enter the platoon. The platoon has to communicate its common goals, as well as the entry requirements, such as role and entry position. In addition, communication is also necessary within the platoon. Before entry, the lead car must ask the members of the platoon, for example, to create a gap at the entry position. After Car E has pulled in, the lead vehicle must ask the other members of the platoon to close this gap again.

Dynamics

Let us now look at the special challenges in the area of dynamics using the example of the entry of a vehicle (Car E) into the platoon.

Goals

The entry of Car E into the platoon may lead to adjustments to the community goals (soft goals) of the platoon. For example, Car E could

bring special sensors that can detect the environment more precisely into the platoon, and thus enable a higher cruising speed for the entire platoon.

Functions and behavior

The entry of Car E can also lead to a change of roles within the platoon. For example, for Car A in its role as leader, the size of the platoon could be limited to four vehicles. For the inclusion of Car E as the fifth vehicle, the leading role must therefore be transferred to one of the other vehicles that supports the corresponding platoon size. However, it could also be that Car A hands over its role as lead vehicle to Car E after entry because Car A will leave the platoon in a short time. Functions such as the coordination of acceleration, braking, and lane change of the platoon then move from Car A to Car E.

Architecture and structure

The entry of Car E changes the internal structure of the platoon, such as the number and order of the participating cars. Depending on the sensor types contained in Car E (e.g., for distance measurement), the interfaces of the other members may have to be adapted. Interface adaptations may also be necessary if sensors are missing or unknown sensor types are used.

Context

The context of the platoon is constantly changing. New vehicles, traffic signs, but also unpredictable obstacles on the road can appear at any time. In addition, new functionalities can appear in context, such as the sensor data of a traffic control system that provide information about the road surface. The platoon must be able to detect these changes fast enough and adapt its behavior accordingly. With the entry of Car E into the platoon, the context changes for the platoon as well as for Car E and the previous vehicles of the platoon. For Car E, the context no longer contains the platoon as a whole, but rather the individual vehicles inside the platoon. For the vehicles of the platoon, Car E now becomes a member of their own association.

Uncertainty

Platoons operate in an open environment and must therefore deal with a high degree of uncertainty and fuzziness. A platoon must be able to deal with road users not yet known at the time of the platoon's design. Road safety must be guaranteed even then. Future vehicles with new features (such as extended information about the environment) should be included in the platoon and their capabilities should be able to be used.

2.5.2 Challenges in the Application Domain “Industry 4.0”

The visions of an adaptable and flexible factory are complex and are described by different scenarios in connection with the Industry 4.0

www.dbooks.org

vision [Plattform Industry 4.0 2017a], [Plattform Industry 4.0 2017b], [Plattform Industry 4.0 2017c].

One scenario frequently described in this context is order-driven production, where the CESs involved in the production of a product (also called modules in the factory) form a CSG (also called production network in the factory) based on the requirements of the product to be manufactured and with the goal of manufacturing the product. The application of the concept described in 2.3.1 results in a production network that is formed to produce a specific production order and is dissolved again after its completion.

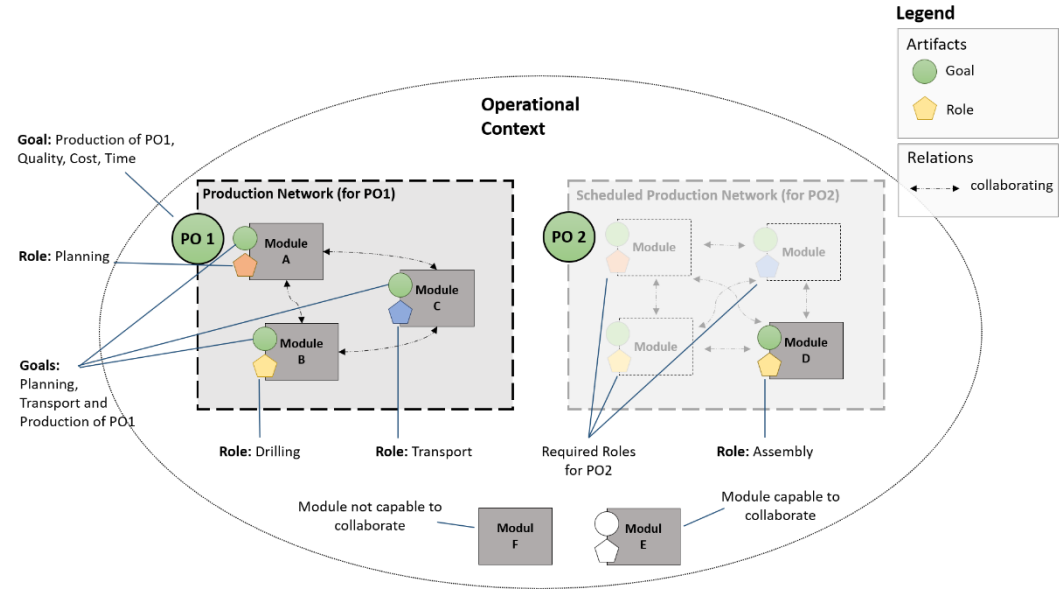


***Fig. 2-6**: Overview of collaboration in order-driven production*

In the example of the adaptable and flexible factory, the interaction of the CESs in the CSG must also be considered and described by means of suitable models in order to serve as a basis for the development of the CESs and to enable collaboration during operation. Figure 2-6 shows such an existing and a planned production network for the processing of two production orders.

To process manufacturing order PO1, a different composition of production modules is required than for manufacturing order PO2 (see Figure 2-6). When the order is received, the modules agree, based on the order information, whether and under which conditions (costs, quality, and time) they can contribute to the production. For the

production of PO1, for example, a collaboration of modules A, B, and C with the roles production planning unit, production station (in the form of a drilling station), and transport device is required, while for the production of PO2, a further function of module D in the role of an assembly device must be added. Even in the adaptable and flexible factory, there are modules that, due to lack of suitable functionality, are not capable of collaborating with other modules (e.g., module F in the figure).

Individual goals of the modules, such as achieving the highest possible throughput, energy-efficient production, or adherence to certain maintenance intervals, must be taken into account when forming production networks and compared with the higher-level and possibly conflicting goals of the production network. The fulfilment of the production order represents the overriding overall goal, which can only be achieved through the individual contributions of the modules involved in production.

In this scenario, it is assumed that in different factories, different modules with heterogeneous system functions or production functions (such as drilling, milling, transport, and assembly) are available for the production of individual customer orders, initially detached from each other. Depending on the shape of the product to be manufactured, different manufacturing functions are required for production. In contrast to platooning, this scenario is mainly characterized by the multitude of very different functions of individual CESs.

The contributing modules must both align their functions with the requirements resulting from the order and communicate their respective contribution to the production to each other in order to jointly determine the feasibility and the sequence of processing. Depending on the functions required, it may be necessary to reconfigure modules before production can start because they cannot perform a required function in the current configuration. Depending on the scope, the reconfiguration can be performed either automatically by the module itself, or by an external actor. The frequency with which individual modules are reconfigured depends on the requirements of the respective production network.

By providing their respective, very heterogeneous functions, the modules assume roles required for the production (such as production planning unit, production cell, assembly station, transport device) and contribute to production within a CSG.

www.dbooks.org

Collaboration

In order to realize the collaboration of the modules for the joint production of a product, numerous challenges have to be met. By combining the very heterogeneous functions of individual modules, it should be possible to manufacture a product that a single module could not manufacture on its own due to its limited possibilities.

Goals

Since each individual module can make only a limited contribution to the overall production, and since these individual contributions must be coordinated for an aggregated overall contribution, achievable intermediate goals (such as progress in the production process that can be achieved by the individual module) must be defined. This requires that the modules have machine-interpretable descriptions for their respective functions and that they exchange these descriptions, as well as metadata (e.g., units used, qualities provided) in the context of communication with other modules through their collaboration functions. From these descriptions, we can derive whether and to what extent a contribution can be made to the production of a product.

Function and behavior

During production, consideration must be given to the fact that the sequence of functions to be performed by the modules varies depending on the product to be manufactured. The information about the sequence of the functions of the modules to be executed is determined based on the production order. While, for example, in the case of a platoon, the functions to be executed for integrating or leaving the platoon are very similar, even with varying vehicles and destinations, in a factory, even when manufacturing very similar products, a geometrically determined, very different sequence of functions may be required in production.

The production sequence as the goal of collaboration and the resulting involvement of the modules must therefore be redefined for each production order. For example, for the production of PO1 and PO2, both functions of module A and module B are required. For PO1, however, it may be necessary for module A to execute its production functions first and module B afterwards, whereas PO2 requires the functions of module B first and then those of module A. PO2 may even require manual reconfiguration by an employee in the factory of module B because a specific tool is required. This reconfiguration must also be considered and provided for in the collaboration.

Architecture and structure

In addition to providing individual system functions, the architecture of factory modules also implements communication with other CSG modules. While communication about platooning targets is done dynamically at system runtime, targets of CSGs and CESs of adaptable

and flexible factories are aligned at configuration time. Due to the heterogeneity of goals, roles, collaboration, and system functions, the ways in which modules are combined to form CSGs are much more complex than in platooning and require intelligent procedures for coordination.

Communication

In the adaptable and flexible factory in particular, and in similar heterogeneous collaborative contexts, it may therefore be necessary to involve people as actors (experts) in the CSG formation process. In addition to the inter-CES communication, opportunities to involve experts in collaboration planning must also be provided.

Dynamics

Goals

The goal pursued by a production network is to fulfil a single production order. The production network comprises all modules that contribute to the production of the corresponding product with their production functions and, if necessary, additional functions such as production monitoring. If, for example, module B fails in its role as a production cell (drilling station) due to an error during the processing of PO1, compensation strategies are required to ensure the fulfilment of the order. In this case, the production network could request the required function from module E, since this module does not belong to any network at this time and has the basic possibility of assuming the required role.

Functions and behavior

The failure of a system function of a module as well as the search for and integration of alternative modules with comparable system functions shows the dynamics with regard to function and behavior. Such a change can result in further adaptations because new transport routes may have to be considered.

Architecture and structure

Every time a module fails or a new module is added to the production network, the architecture and structure of the network also change.

Context

Furthermore, production networks must be able to deal with changes in their context. For example, the delivery of required materials by external suppliers in the context of PO1 could change.

Uncertainty

While in platooning, a vehicle with the role of "following vehicle" leaving the platoon can be assumed to be an everyday occurrence and usually does not prevent the joint achievement of goals, the failure of modules in the production network can mean that the order cannot be fulfilled. These and other forms of uncertainty must also be taken into account during the design of individual modules of the adaptable and flexible factory.

www.dbooks.org

2.6 Concepts and Methods for the Development of Collaborative Embedded Systems

2.6.1 Enhancements Regarding SPES2020 and SPES_XT

To meet the requirements for the development of collaborative embedded systems, new methods have been developed in CrESt. These methods were classified according to their contribution to the taxonomy of challenges (see Figure 2-4). Some methods can be classified into the taxonomy several times because they offer solutions for different challenges.

Process building blocks

The "Process Building Block Framework" was used to document the methods (see [Pohl et al. 2016]). This framework allows systematic documentation of how artifacts are created and processed in the development process. Each "process building block" has a well-defined input (e.g., models, etc.) and output (models, analysis results, etc.). Input and output can be further restricted by pre- and post-conditions and are assigned to the SPES viewpoints. Process building blocks can be connected to each other via relationships and thus provide a mapping to the desired development process.

SPES viewpoints and cross-cutting topics

In SPES2020 and SPES_XT, a framework for the creation of a system model was developed. The models are organized in four viewpoints: requirements viewpoint, functional viewpoint, logical viewpoint, and technical viewpoint. In addition, the framework includes special models for cross-cutting topics such as safety, variability, and validation. For the description of CESs in a system model, this framework has been extended in CrESt. The existing viewpoint structure was retained. Existing models were extended and a number of new model types were defined. These extensions are used, among other things, to describe a CSG and its relationships to CESs — for example, with respect to goals and functions (see Figure 2-7). The two main classes of the taxonomy (Figure 2-4) "Collaboration" and "Dynamics" impact all four viewpoints. Therefore, they have to be considered in the models of all viewpoints. In addition to the extensions of existing viewpoints, specific model types have been defined that consider collaboration and dynamics and cannot be

assigned directly to existing viewpoints. They are orthogonal to the

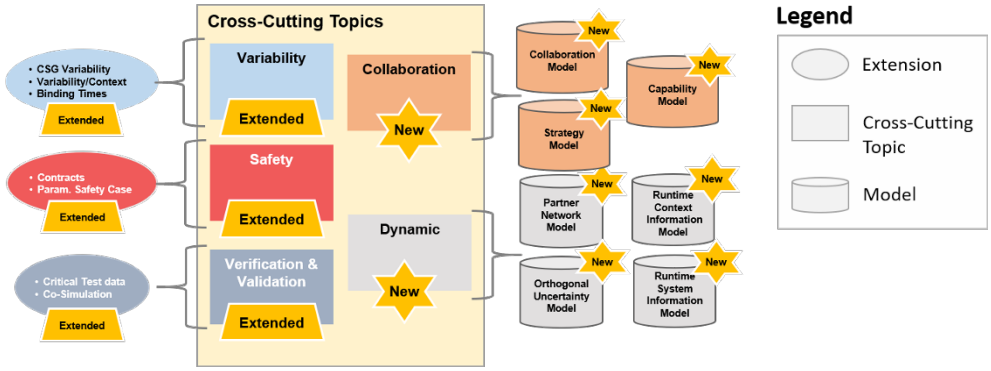


***Fig. 2-7**: CrESt framework (part1)*

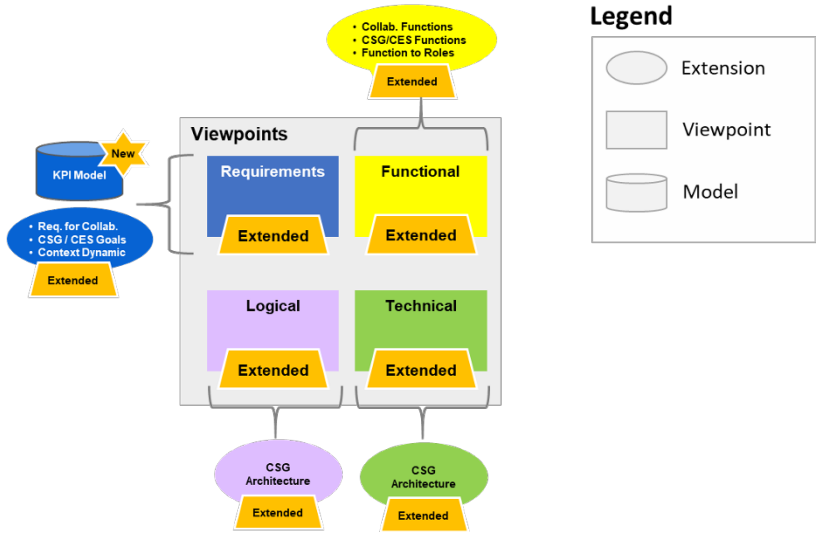


***Fig. 2-8**: CrESt framework (part2)*

viewpoints and have cross-connections to several viewpoints. Examples are models that describe collaboration strategies in a CSG or information about dynamics at runtime (see Figure 2-8).

2.6.2 Collaboration

In order to support the development of CESs and CSGs in terms of collaboration, the methodological toolbox of SPES, including the modeling approach contained therein, was extended in CrESt. A list of

www.dbooks.org

methods developed to support collaboration can be found in the appendix of this chapter[6].

Goals

Goal-Oriented Requirements Language

The Goal-Oriented Requirements Language (GRL) [Daun et al. 2019] can be used to model the common goals of a CSG and the relationships to the individual goals of the CES members. With the help of this formal description, the necessary skills and key performance indicators (KPIs) of the CSG members, whose interaction in the context of a collaboration makes the achievement of the common goals possible, can be derived.

Language for partner network models

In order to analyze the individual goals of potential CSG members or their development organizations, CrESt defined a suitable language for partner network models.

Combination of different variability models

In order to illustrate the variability or configurability of a CSG based on the configuration possibilities of its members, CrESt results allow for the combination of different variability models.

Based on these extensions of the modeling framework, specific methods were developed to achieve the goals of a collaboration. Thus, it is possible to determine, at runtime, whether or not a collaborative goal can be achieved in the current CES constellation with the CES capabilities currently available. The possibility to achieve a common goal by making possible adjustments to the participating CESs is also taken into account. For example, this approach can be used to determine, for an adaptable and flexible factory, whether it is possible to produce a product with the required quality. If not, the approach allows a check to determine whether a suitable (re-) configuration of the modules can make that production possible. Further details can be found in Chapter 6.

In order to achieve a common goal of a CSG, it may be necessary for individual members to adapt the individual goals they pursue in order to subordinate them to those of the group. For example, in order to reduce their own fuel consumption by participating in a platoon, all participating vehicles must adapt their speed to the speed of the platoon. With the help of CrESt methods, suitable strategies can be derived and verified at runtime to optimally achieve both the common goals of a CSG and the goals pursued by the members of the collaborating CESs (for details, see Chapter 9 and Chapter 10).

[6] The CrESt results are available on request. See: https://crest.in.tum.de/ (website available in German only),

Functions and Behavior

The modeling of functions has been extended to support the modeling of CESs in the framework. Two types of functions can now be distinguished.

Modeling of collaboration functions

The first type of function is the collaboration functions necessary for the collaboration of CESs in a CSG and the system functions of a CES that serve to achieve the system goals. To enable the dedicated consideration of collaboration functions in particular, appropriate modeling approaches were provided in order to ultimately enable collaboration at runtime and the associated systematic coordination of different functionalities in a CSG. When designing a CSG, it is now possible to specify which collaboration functions are necessary for the participation of a CES in a CSG.

Goals and roles in a collaboration

In addition, the conceptual relationships between system and CSG functions, as well as, for example, goals and roles in a collaboration, were worked out. For a CSG, we can specify which roles the individual CES members can take on and which functions they have to provide to the CSG for this purpose. In a platoon, for example, the lead vehicle must be able to plan and execute lane changes for the entire platoon. By means of collaboration functions, system functions such as acceleration and braking must be orchestrated in a suitable way, so that, for example, the entry and exit of a vehicle into and from a platoon is made possible. A detailed description of function modeling can be found in Chapter 4 and Chapter 5.

Describing CSG behavior

Additionally, the framework has been extended to formally describe the behavior of a CSG through contracts and scenarios at design time (see Section 8.4). Furthermore, approaches including suitable tools were developed to analyze the behavior of a CSG by co-simulation. Details can be found in Chapter 12 and Chapter 13. The confidence of the CES members in the behavior of the other members plays an important role in the creation of a CSG. In CrESt, an approach was therefore developed to build up mutual trust in the behavior of CES members, for instance within a platoon, with the help of digital twins (see Chapter 14).

Architecture and Structure

Support of virtual CSG characteristics

The goal-oriented requirements models are used in CrESt to derive supporting architectures of CESs and CSGs. In addition, the architecture modeling in the framework has been extended to support the virtual characteristics of a CSG. This means that all components of a CSG architecture are realized by components of the participating

www.dbooks.org

CESs. The design of a CSG is therefore described at two levels. At the development stage, the architecture is defined at an abstract level with the help of reference architectures developed in CrESt (for example, in the context of standardization). A detailed description of this approach can be found in Chapter 3 and Chapter 5. At runtime, the abstract architecture is instantiated into a concrete architecture. For example, only the framework conditions for a platoon are specified at design time. At runtime, a platoon then consists of a defined number of concrete vehicles.

Communication

CESs must be able to communicate with the different partners both within a CSG and in their environment. For example, in a platoon, the following vehicles must be able to be instructed by the lead vehicle to form a gap for the entering vehicle at the given position.

Semantic interoperability

In CrESt, an approach has been developed to achieve semantic interoperability between different and changing communication partners regarding the exchanged (possibly complex) information by means of ontologies. For example, it is important to exchange information about the specific capabilities of the individual CES members. The CrESt framework was therefore extended by a formal description of the capabilities of a CES (for details, see Section 6.3 and Chapter 7).

Safety contracts

Safety contracts must also be communicated at runtime for safety-critical systems. In CrESt, a corresponding method has been developed for this purpose. (see Section 8.4). Furthermore, suitable communication patterns were defined for the communication of CESs in a CSG and made available on the basis of the Coaty middleware framework [Coaty]. A detailed description can be found in Section 10.6.

2.6.3 Dynamics

With regard to the problem dimension dynamics, both the modeling of CESs and CSGs and the methodological toolkit for developing these systems have been expanded. The appendix of this document contains a list of the methods developed in CrESt for this dimension[7].

[7] The CrESt results are available on request. See: https://crest.in.tum.de/ (website available in German only).

Goals

Strategies for individual and community goals

With the approaches developed in CrESt, community goals can now be negotiated dynamically at runtime. The decisions made at runtime are based on strategies for ensuring that individual and community goals are achieved as well as possible. Such strategies also serve to plan and enable adaptation at runtime based on the achievement of goals. CrESt provides methods for deriving appropriate strategies and operationalizing the adaptation (see also Chapter 9).

Commitments

In order to achieve the common goals of a CSG, all CES members must fulfill their commitments. Therefore, CrESt has developed methods for assessing the risk and impact of erroneous behavior of a CES and for ensuring that the goals are met in this case as well. In the use case of the adaptable and flexible factory, for example, the failure of one module must not lead to serious damage to the factory workers and the other modules. A prerequisite for this is a method for a goal-based review of CESs at runtime. These methods are described in Section 10.2 and Section 8.3.4.

Functions and Behavior

Safety-critical analyses

Both CESs and CSGs change their behavior dynamically at runtime. In the case of a CSG, for example, this can be done by CES members joining and leaving the CSG. This dynamic behavior makes it difficult to perform safety-critical analyses completely at design time. Therefore, methods are made available in the framework where parts of the security analysis can be shifted to runtime. In order to execute these parts with acceptable effort at runtime, corresponding preparatory work at design time is necessary. Analyses with regard to risks, errors, and uncertainties can thus be analyzed in a model-based manner at design time and combined into modular safety checks that are evaluated at runtime (see Chapter 8 and Chapter 3). In addition, argument-based and contract-based approaches based on behavioral models that allow a semi-automated or fully automated safety demonstration at runtime have been developed (Section 8.4). A model-based approach to risk analysis supports safety engineers in assessing the safety of newly configured systems at runtime. Problems arising from adjustments to systems at runtime can be identified by predictive simulation under certain circumstances. A detailed description can be found in Section 10.3.

Monitoring methods

In addition, CrESt also provides suitable monitoring methods that monitor both functional behavior and time behavior.

www.dbooks.org

Architecture and Structure

Dynamic architectures

In CrESt, an approach was developed for deriving a dynamic architecture by considering the corresponding architectures for different context situations. Dynamic architectures of CSGs can also be designed using reference architectures from the building set. At the development stage, the architecture is defined only at an abstract level (for instance, in the context of standardization). The concrete CESs or the number of CESs that make up the architecture at runtime are not known at this time. Only at runtime is this abstract architecture instantiated into a concrete architecture (for details, see Chapter 3 and Chapter 5. During runtime, this concrete architecture can change again and again as the members of the CSG change — that is, when they enter or leave the system.

Dynamic binding

In CrESt, approaches from software product line development were used to enable the dynamic binding of components at runtime and to analyze possible architectures at development time with regard to their variable—that is, potentially dynamic—components. This approach is detailed in Chapter 5. A CES is often integrated in different CSGs, which poses special challenges for variability modeling at design time, since short-term change requests to a CES are often implemented only for a specific configuration in a single CSG. In CrESt, methods have been developed to merge these changes into a single CES configuration with the current version of the CES product lines fully automatically (see Chapter 18 for more details).

Co-simulation methods

Testing of a designed CSG is made more difficult by the fact that a large number of different CES combinations are possible. In order to test a large number of scenarios during development, methods for co-simulating the real world with the virtual world in CrESt were developed. Using evolutionary test methods, the critical situations of a system can be identified and the quantity of test cases can be reduced to these situations.

Context

Context-sensitive variability models

CESs operate in a constantly changing environment to which they have to adapt their behavior. In CrESt, approaches have been developed to support the systems in adapting and using context information. The creation of context-sensitive variability models facilitates the search for a valid CSG configuration for a changed context.

Digital twins

Another approach combines the use of digital twins with predictive simulation using the perceived context to find the optimal

configuration for each situation (Section 3.2, Chapter 15, and Section 10.3).

Runtime-specific context models

In addition, methods have also been developed to observe and evaluate the effects of context changes on the system and its behavior at runtime (see Section 8.3.1). These are based on the modeling of runtime-specific context models. The CrESt framework now also supports sufficient testing of adapting systems in a dynamic environment. Further details can be found in Chapter 6.

Uncertainty

Identification of uncertainty types

CESs operate in an open and dynamic environment. They are developed independently of each other and can combine to form different constellations at runtime. This significantly increases the complexity regarding potential uncertainties that can occur at runtime. In CrESt, methods have been developed to systematically identify the different types of uncertainties (e.g., regarding collaboration, data quality, sensor perception, information exchange) at design time (see Chapter 7). For the systematic documentation of the uncertainties identified, a model-based approach was developed in which the uncertainties are described orthogonally—that is, in separate models with uncertainty-specific model elements—and are related to various system or context models.

For an adaptable and flexible factory, for example, this allows uncertainties that could disrupt the production process or lead to a production stop to be analyzed and documented. The uncertainties identified can then be linked to the models of the individual machines and the model used to describe the production process. For a platoon, this method can be used to identify and model uncertainties such as incompleteness and ambiguity with regard to the information exchanged between vehicles on the driving environment (for details see Section 7.3.1).

AI-based techniques for data-driven components

Another method developed in CrESt aims at identifying and handling uncertainties that may arise from the use of data-driven components—that is, AI-based techniques—for the evaluation of environmental data (see Section 7.3.2). For this purpose, the quantification of the uncertainty regarding the output of data-driven components (e.g., the recognition of a traffic sign) at runtime is enabled. This serves to ensure that data-driven components whose behavior cannot be completely predicted at development time meet safety-critical requirements during operation.

www.dbooks.org

2.7 Conclusion

Within the CrESt project, important concepts of collaborative embedded systems were identified. From the resulting specific challenges, a number of key features (such as goals, communication, uncertainty) were developed. The methodological building blocks developed, as well as the extensions of existing building blocks, focus on addressing these challenges and were assigned to the main features.

Restrictive assumptions

A specific, somewhat more restrictive system concept was deliberately chosen as the basis for the work. On the one hand, the assumption was made that a CES collaborates in at most one CSG at any given time. On the other hand, hierarchical CSGs (i.e., system networks of system networks) were excluded from the analysis. For many use cases, including those considered in the project, these assumptions are quite practical. Future work in this topic area should look more closely at these limitations.

Integration of AI technologies

Increasingly, methods of artificial intelligence (AI) are being used in embedded systems. The AI methods (for example, machine learning, deep learning, data analytics, semantic technologies) are as diverse as their applications. These range from the analysis and classification of existing situations to the interpretation and evaluation, diagnosis and prognosis, and the creation of proposals for action and independent action in the sense of autonomous systems. The use of AI technologies makes it possible to process incoming information appropriately and to adapt to changing conditions at runtime.

A central challenge for the integration of AI technologies in CESs and CSGs is to guarantee the essential functionality and quality characteristics of the systems. In general, the behavior of AI methods cannot be completely determined at development time. Therefore, it is unclear which adaptations the systems make at runtime and in what way this influences the collaboration and dynamics of the CESs and CSGs. An interesting question here is whether and how the necessary conceptual development of the CSG level can be replaced by the use of AI methods at runtime.

Novel challenges

Furthermore, the integration of AI components in the context of uncertainties leads to novel effects and challenges that have to be considered as early as development time. These include, for example, data that is not 100% trustworthy (i.e., data with undetected systematic deviations or fuzziness), non-deterministic behavior, runtime variances, malicious misinformation, and commands from

outside the system boundaries. These uncertainties affect the knowledge gained from AI components. This and the dynamic adaptability create completely new challenges for the development and quality assurance of embedded systems.

The secured integration of powerful AI technologies in CESs and CSGs marks a decisive development step for future collaborative systems. The necessary extensions of the design methodology would have to be investigated in future projects.

2.8 Literature

[Bandyszak et al. 2020] T. Bandyszak, M. Daun, B. Tenbergen, P. Kuhs, S. Wolf, T. Weyer: Orthogonal Uncertainty Modeling in the Engineering of Cyber-Physical Systems. In: IEEE Transactions on Automation Science and Engineering, IEEE 2020.

[Bresciani et al. 2004] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos: Tropos: An Agent-Oriented Software Development Methodology. In: Autonomous Agents and Multi-Agent Systems 8, 2004, pp. 203–236.

[Broy and Stolen 2001] M. Broy, K. Stolen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement, Springer 2001.

[Broy 2010] M. Broy: A Logical Basis for Component-Oriented Software and Systems Engineering. In: The Computer Journal, vol. 53, no. 10, 2010, pp. 1758-1782.

[Broy and Schmidt 2014] M. Broy, A. Schmidt: Challenges in Engineering Cyber-Physical Systems. IEEE Computer, 47(2), IEEE 2014, pp. 70-72.

[Broy et al. 2020] M. Broy, W. Böhm, M. Junker, A. Vogelsang, S. Voss: Praxisnahe Einführung von MBSE – Vorgehen und Lessons Learnt, White Paper, fortiss GmbH, 2020 (available in German only).

[Coaty] Coaty.io: Coaty Developer Guide. https://coatyio.github.io/coaty-js/man/developer-guide/, accessed on: 07/14/2020.

[Damm and Vincentelli 2015] W. Damm, A. S. Vincentelli: A Conceptual Model of System of Systems. Second International Workshop on the Swarm at the Edge of the Cloud, ACM 2015.

[Daun et al. 2019] M. Daun, V. Stenkova, L. Krajinski, J. Brings, T. Bandyszak, T. Weyer: Goal Modeling for Collaborative Groups of Cyber-Physical Systems with GRL: Reflections on Applicability and Limitations Based on Two Studies Conducted in Industry. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, 2019.

www.dbooks.org

[France and Rumpe 2007] R. France, B. Rumpe: Model-Driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering (FOSE'07), IEEE 2007, pp. 37-54.

[Grosz 1996] B. J. Grosz: Collaborative Systems. In: AI Magazine, vol 17, no 2, 1996.

[Horkoff et al. 2019] J. Horkoff, F. Başak Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, P. Giorgini: Goal-Oriented Requirements Engineering: An Extended Systematic Mapping Study. In: Requirements Engineering 24, 2 (June 2019), 2019, pp. 133–160.

[Keet et al. 2013] C. M. Keet, W. Dubitzky, O. Wolkenhauer, K.-H. Cho, H. Yokota: Open World Assumption. In: Encyclopedia of Systems Biology, Springer, New York, 2013.

[Lamsweerde 2000] A. v. Lamsweerde: Requirements Engineering in the Year 00: A Research Perspective. In: Proceedings of the 22nd International Conference on Software Engineering, Invited Paper, ACM Press, June 2000.

[Maier 1998] Mark W. Maier: Architecting Principles for Systems-of-Systems. In: Systems Engineering 1(4), 1998, pp. 267-284.

[Plattform Industrie 4.0 2017a] Plattform Industrie 4.0: Application scenario in practice: order-controlled production of a customised bicycle handlebar. BMWi, Berlin, 2017.

[Plattform Industrie 4.0 2017b] Platform Industrie 4.0: Aspects of the Research Roadmap in Application Scenarios. BMWi. https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf; accessed on 04/04/2020.

[Plattform Industrie 4.0 2017c] Plattform Industrie 4.0: Fortschreibung Anwendungsszenarien der Plattform Industrie 4.0. BMWi: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html; accessed on 04/08/2020.

[Pohl et al. 2012] K. Pohl, H. Hönninger, R. Achatz, M. Broy (Eds.): Model-Based Engineering of Embedded Systems: The SPES2020 Methodology, Springer, Heidelberg/New York, 2012.

[Pohl et al. 2016] K. Pohl, M. Broy, H. Daembkes, H. Hönninger (Eds.): Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES2020 Methodology, Springer, Heidelberg/New York, 2016.

[SafeTRANS 2019] SafeTRANS e.V.: Safety, Security, and Certifiability of Future Man-Machine Systems, 2019.

[Selic 2003] B. Selic: The Pragmatics of Model-Driven Development. In: IEEE Software, 20(5), IEEE 2003, pp. 19-25.

[Sha et al. 2008] L. Sha, S. Gopalakrishnan, X. Liu, Q. Wang: Cyber-Physical Systems: A New Frontier. In: 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008), 2008, pp. 1–9.

2.9 Appendix

In the CrESt project, methods and building blocks for modeling collaborative systems and system networks were developed. The documents containing a detailed description of the project results can be requested via the project website (https://crest.in.tum.de/, website available in German only).

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Birthe Böhm, Siemens AG
Carmen Cârlan, fortiss GmbH
Annelie Sohr, Siemens AG
Stephan Unverdorben, Siemens AG
Jan Vollmar, Siemens AG

3

Architectures for Flexible Collaborative Systems

Collaborative systems are characterized by their interaction with other systems in collaborative system groups in order to reach a common goal. These systems interact based on fixed rules and have the ability to change structurally, if necessary. Changes in the collaboration are usually triggered from outside and are time-discrete with a rather wide time scale. The architectures of these systems and system groups must support flexibility and adaptability at runtime while also ensuring specific qualities, although these changes and their consequences cannot be fully foreseen in all combinations at design time.

In order to enable knowledge preservation and reuse for the design of system architectures for flexible collaborative systems and system groups, we present a method for designing reference architectures for systems and system groups. For this approach, we present an example of a reference architecture for an operator assistance system. To adequately consider safety requirements during the design, we further introduce a method which adapts safety argumentation for flexible collaborative systems to changes in their specification or operating context.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_3

3.1 Introduction

Designing architectures for flexible collaborative systems and their system groups is still a challenge due to the novelty of these systems and a lack of proven methods that address their specific requirements [Böhm et al. 2018]. This applies in particular to the design of system groups and the systems collaborating within these groups.

Characteristics of flexible collaborative systems

Flexible collaborative systems assume a fixed collaboration that adheres to a fixed set of rules. Changes are usually triggered not from the system itself but, for example, by an operator of this system. These changes are not as frequent as in dynamically coupled or adaptive systems. Typical examples of flexible collaborative systems are adaptable and flexible factories.

In Section 3.2, we provide a method for designing reference architectures for collaborative embedded systems (CESs) and collaborative system groups (CSGs). Such reference architectures can then be used as blueprints for deriving system architectures for specific systems. In addition, they can be used to design specific CSGs and collaborating CESs at an interface level to allow for independent design and development of the CESs and CSGs but enable their collaboration. We then apply this approach to adaptable and flexible factories, and briefly present the resulting high-level logical reference architecture. This overview is detailed in Section 3.3 by applying the approach to one of the CESs identified, a simulation-based operator assistance system.

For numerous CESs and their CSGs, safety requirements are crucial and must be guaranteed. Our proposed safety case modeling approach in Section 3.4 supports the execution of automatic consistency checks between the safety case model and the system architecture. This approach can be used to prove that the architecture of a system satisfies the required safety properties. It ensures that, in the event of changes to the system specification or the operating context, the logical architecture still fulfills the safety requirements.

Finally, in Section 3.5, we provide conclusions and give an outlook on future work.

3.2 Designing Reference Architectures

A typical approach for designing architectures for systems starts with eliciting specific requirements. This step is followed by identifying

www.dbooks.org

functions needed. Based on these functions, we create a logical architecture and, finally, a problem-specific technical architecture. This procedure must be repeated from scratch for each specific system. Therefore, in particular for organizations that frequently design similar systems, reuse of existing solutions promises a reduction in effort and the possibility to make experiences and knowledge available to future projects or even across organizational borders. Various reuse approaches can be classified. For example, VDI/VDE 3695 defines, among other things, reference models or architectures as one possible way of enabling reuse of artifacts within the engineering of systems [VDI/VDE 3695 2010].

Reference architectures are a reuse approach for organizations that expect to build similar systems in the future and already have good knowledge of these systems. They are used as blueprints for future systems and may be adapted for specific systems. In addition, reference architectures may be also applied when designing specific CSGs (e.g., for the adaptable and flexible factory) to define the necessary roles, system, and collaboration functions of CESs but also protocols, data structures, etc. to enable collaboration within this CSG. Different organizations may subsequently use this reference architecture to design CESs which may collaborate in these CSGs.

In this section, we present a method for designing reference architectures for CESs and CSGs. In addition, we give a short insight into a reference architecture for adaptable and flexible factories. This reference architecture is based on a general reference architecture for CESs and CSGs.

Definition of reference architecture and architecture framework

A reference architecture is defined as "the outcome of applying the architectural framework to a class of systems to provide guidance and to identify, analyze and resolve common, important architectural concerns. A reference architecture can be used as a template for concrete architecture of systems of the class" [Lin et al. 2017]. Complementing this, an architecture framework is defined as "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders" [ISO/IEC/IEEE 42010 2011]. The SPES_XT modeling framework (see Chapter 2) is an example of such an architecture framework and is used in the following for designing reference architectures as well as system architectures.

3.2.1 Method for Designing Reference Architectures

The general procedure for designing reference architectures and deriving system architectures from reference architectures is shown in Figure 3-1. While a reference architecture is created only once, numerous system architectures can be derived from a single reference architecture. The transitions between the viewpoints in Figure 3-1 show the general procedure for designing reference and system architectures.

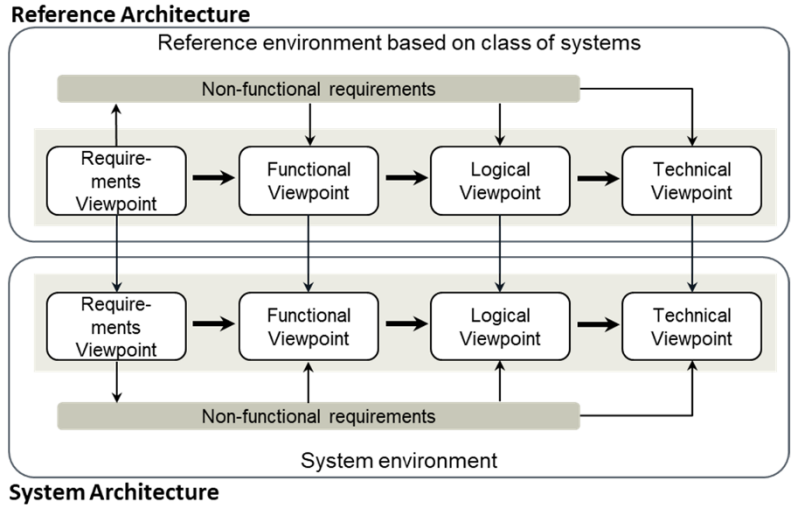


Fig. 3-1: *General approach for designing reference and system architectures*

Non-functional requirements

In addition, the role of non-functional requirements (e.g., requirements related to safety), which are elicited in the requirements viewpoint, is highlighted. In some cases, these requirements cannot be assigned to single functions or to logical or technical components and should therefore be revised regularly during the design of reference as well as system architectures — this is indicated by the arrows related to the non-functional requirements in the figure above. In Section 3.4, we provide a method for integrating safety cases into reference or system architectures to provide an approach for safety-related requirements.

Finally, in Figure 3-1, the arrows from the reference architecture viewpoints to the system architecture viewpoints indicate the reuse of design results for designing system architectures. However, it may be necessary to adapt or complement the reference architecture content.

www.dbooks.org

Scope definition

As a first and critical step within the requirements viewpoint for defining reference architectures, we define the scope of systems for which the reference architecture will be defined. This means that we need to forecast the future systems for which we want to use the reference architecture as a blueprint.

Key design decisions for reference architectures

Next, we determine which kind of reference architecture we want to design. There are several key design decisions that have to be taken, for example:

- *Coverage*: Reference architectures can, for example, cover a common core of all considered systems, offer combinable and reusable building blocks, or provide a solution that will cover all requirements of all considered systems and is then tailored to fit to one specific system.
- *Extensibility*: Reference architectures may, for example, allow white box extensibility, which means that its components can be fully adapted. On the other hand, only black box reuse that does not allow any internal modifications may be allowed. Other forms include grey box reuse, which is a mixture of both.
- *Granularity*: The level of granularity of the reference architecture must also be decided. The goal is to be as detailed as possible while still covering the future system architectures for the intended set of systems. A reference architecture may, for example, define only interfaces of systems or components or provide a full detailing of all systems.
- *Viewpoints*: Consequently, the reference architecture may define views of the requirements viewpoint only, or also comprise views of the functional, logical, technical, and other viewpoints. While a reference architecture that covers all viewpoints would appear to be the best option, it also allows less changeability or requires more effort if there are frequent changes.

These key design decisions mainly depend on the similarity of the set of systems and their requirements.

Further requirements elicitation considers the scope of systems

Subsequently, further requirements are elicited for the reference architecture based on the decisions made above. In addition, even requirements of the set of selected systems that are not implemented by the reference architecture may have to be considered to prepare their later implementation. For collaborative systems in particular, the CSG must be considered as well as the CESs — for both CSG and CES design. This results from the general concept described in Chapter 2. If a CES is to contribute to different CSGs, all relevant CSGs have to be involved.

Functional architecture

On this basis, we then extract the necessary functions of our reference architecture while also considering the collaboration and system functions for both CSGs and CESs. It is important to keep the relations between requirements and functions, and further on, to logical and technical components, as traces. These traces allow us to check, for example, whether all requirements are implemented by functions or logical and technical components. Vice versa, in the case of changes to the technical solution, the traces also enable us to check whether all requirements are still fulfilled.

Logical architecture

Based on the functional architecture, we create a logical architecture for the set of selected systems. Within this logical architecture, the CSGs are usually logical components and are composed by the CESs.

Technical architecture

Finally, a technical reference architecture may be created. Since CSGs are virtual, the collaboration and system functions have to be implemented by the CESs. For all architecture viewpoints, it is crucial to document design decisions and trace the relationships between the different viewpoints and between the elements in the viewpoints. We then refine any viewpoints as far as possible.

Deriving system architectures from reference architectures

Once the reference architecture is created, we can use it to derive system architectures for future systems. Again, we need to elicit requirements but now for a specific system we want to build. We then compare these requirements with the requirements for our reference architecture and identify similarities as well as differences. Subsequently, we assess these similarities and differences while keeping in mind the parameters for our reference architecture. By using the traces between all architectural components, we can then customize the reference architecture by following the traces and adjusting the elements with divergent or refined requirements — if our extensibility concept permits these adaptions. In addition, we have to integrate new requirements which have not been considered in the reference architecture but are needed for the specific system [Unverdorben et al. 2019].

Example 3-2: Using a Reference Architecture as a Template

Imagine a reference architecture for a factory which includes a requirement to display all alarm data to operators to allow them to recognize critical situations and ensure smooth production. However, for one specific factory, the data will be analyzed first to identify critical situations and only decision-relevant data will be displayed to the operators.

www.dbooks.org

Since just one requirement has changed, we still want to use the reference architecture for this factory. Therefore, we identify the changed requirement in the reference architecture and follow the traces to related requirements (e.g., alarms will be displayed in a flat list), functions, and logical and technical components. In our example, we find that all requirements dedicated to the data collection are still applicable and the related functions and logical and technical components can remain unchanged. However, the requirements that address data preparation for the operator must be replaced by, firstly, data analysis and, secondly, an adapted user interface for the operator. This affects the related functions but also the logical and technical components. For example, an additional data analysis function is introduced which is assigned to a logical data analysis component. In the technical solution, this logical component is realized by an additional software component.

Note that any changes to the original reference architecture during derivation of a system architecture must be reflected on carefully since they may indicate improvements for the reference architecture. Thus, continuous feedback from system architecture design to reference architecture design is important for keeping the reference architecture up to date. In the case of changes to the reference architecture, there must also be an update concept for existing systems based on a prior version of the reference architecture.

To use the method described above successfully, tool support for modeling reference and system architectures is useful. [Böhm et al. 2020] introduces a modeling tool which implements this method.

3.2.2 Application Example: Reference Architecture for Adaptable and Flexible Factories

For adaptable and flexible factories, we created a reference architecture using the method described above. The focus is on core requirements and the reference architecture must cover the requirements, functional, and logical viewpoints. Since we want to be independent from any specific technical solution, the objective is not a technical reference architecture.

Requirements for the adaptable and flexible factory

The adaptable and flexible factory was already introduced in Chapter 1. In order to extend the requirements for such a factory, we used the application scenarios described in [BMWi 2017a] and [BMWi 2017b] as a basis: the main goal of the factory is to produce products. Incoming product orders must be analyzed in terms of required capabilities and compared with available capabilities within and, optionally, across factories (see also Section 6.4.2). The factory might need to reconfigure its production and, eventually, produces the

product. Besides this basic production process, we assumed that a need for high capacity utilization and guaranteed delivery dates requires production planning. Other goals of the factory are optimization of production, integrated maintenance, collaboration in marketplaces, and continuous development of its product portfolio.

In addition to the application scenarios, requirements arose from the use cases described in this book and the concepts presented in Chapter 2 as guiding principles. On this basis, we designed a general reference architecture for CESs and their CSGs, which not only considers the general concepts but also refines, for example, collaboration and system functions and, subsequently, the logical architecture.

We then created our reference architecture for adaptable and flexible factories. Figure 3-3 shows a basic diagram of the logical reference architecture which presents the CSGs identified, which are derived from the base CSG at the top.

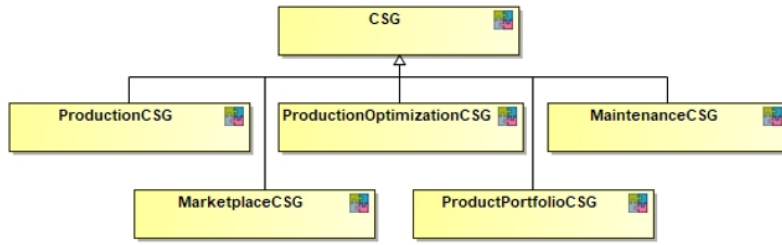


Fig. 3-3: *Refinement of CSG for adaptable and flexible factories*

The CSGs within the reference architecture for adaptable and flexible factories have the following goals and define, accordingly, the following functions:

- *ProductionCSG*: The goal of this CSG is the manufacture of a product specified within a production order. For this purpose, it realizes functions for analyzing incoming product orders with respect to producibility and additional constraints such as delivery dates, price, etc. It also contains functions, for example, for maintaining a production plan for this product, tracking the production, and collecting data for operation control.
- *ProductionOptimizationCSG*: The main goal of this CSG is to optimize the production of the factory. Therefore, it realizes operator support functions—for example, detecting bottlenecks, failures, or unused capacities in production—and deduces measures based on these observations. A close interaction between this CSG and the operator is crucial and may be realized

www.dbooks.org

by an operator assistance system as part of this CSG. This CES is described in more detail in Section 3.3.

- *MaintenanceCSG*: In order to keep the factory productive and in a good state, this CSG defines functions related to preventive and reactive maintenance, as well as maintenance planning and implementation.
- *MarketplaceCSG*: This CSG ensures collaboration between adaptable and flexible factories by offering production capabilities available in the factory and requesting external capabilities via marketplaces.
- *ProductPortfolioCSG*: The goal of this CSG is the continuous development of the factory in order to, for example, reach a high capacity utilization. For this purpose, it combines functions for analyzing missing production functions according to recent product orders, detecting possible improvements (e.g., based on current bottlenecks), and suggesting corresponding measures, etc.

For these CSGs as well as for CESs within the adaptable and flexible factory, the logical architecture was detailed further.

Application to demonstrator

We also used the reference architecture for a factory model demonstrator to derive a specific logical system architecture and to define a technical architecture on top. This pilot showed that the reference architecture is a good basis for deriving system architectures, provided that the underlying general concept is applicable.

3.3 Reference Architecture for Operator Assistance Systems

In Subsection 3.2.2, we identified a CSG for production optimization for adaptable and flexible factories. A central CES contributing to this goal is an operator assistance system. It manages the collaboration of the various CESs in the CSG and offers an interface to the human operator. The CESs being handled by the operator assistance system comprise production machines providing data and they are controlled by the operator, planning and management tools, and additionally model- and data-based evaluation services such as simulation and optimization. These CESs must be combined dynamically in a context- and situation-specific manner. In this section, we now want to take a deep dive into a technical reference architecture for an operator assistance CES.

3.3.1 Simulation-Based Operator Assistance

Simulation can help to optimize production

It is a challenging task to operate adaptable and flexible systems, such as production plants in discrete manufacturing and process industries or connected infrastructure systems such as energy and water grids. The need for more flexibility in operation grows with a higher variety of products, smaller lot sizes, and fluctuating markets. Despite an increasing degree of automation, there are still many decisions to be made by human operators in a short time that target various aspects such as cost, time, and quality. Specific data- and simulation-driven operator support applications can help to handle the task [Boschert et al. 2018], [Rosen et al. 2018]. A digital twin, that is, a virtual replica of the physical system, connects data from different sources and models from different hierarchies. It can form the core of intelligent operator assistance systems [Rosen et al. 2019].

Reference architecture as enabler for low-code assistance system development

Today, integrating simulation and digital twin approaches into operation support for complex systems is still a time-consuming and resource-intensive, typically customer- and project-specific task. You need automation, software, simulation, and domain experts to do this. Therefore, we want to present a technical reference architecture that can support the development of such assistance systems. By using the reference architecture, operator assistance systems can be easily realized on a low-code and low-modeling base and development time can be reduced significantly.

Assistance systems need to be very flexible

One of the main challenges for the development of an operator assistance CES is that it requires a high degree of flexibility: the CES provides different applications such as virtual monitoring and short-term prediction and optimization on different levels such as machine, line, and factory level, and can run in different situations such as normal operation and failure situations. This imposes the need for flexible, situation-specific collaboration of calculation modules and multiple use of data and models.

The concept of a reference architecture for an operator assistance CES will be outlined in the following. For more details, the reader is referred to [Zhou et al. 2019].

3.3.2 Design Decisions

Reference architecture contains execution core and collections of basic elements

We make the following key design decisions for the operator assistance reference architecture:

- *Scope*: We consider simulation-based assistance systems for the operation of adaptable and flexible factories.

www.dbooks.org

- *Coverage*: We cover a common core with generic metamodels and an execution engine to run configurable workflows of evaluations and an extendible collection of re-usable data interfaces, evaluations, and user interface (UI) elements.
- *Extensibility*: The common core is limited to black box reuse in order to guarantee interoperability of services in arbitrary workflows, for different assistance functions, across different plants, and over time. Full white box extensibility is provided for the collections of data interfaces, evaluations, and UI elements.
- *Granularity and viewpoints*: A detailed technical architecture is set up since we aim to implement the architecture as a software framework for the future development of operator assistance systems.

3.3.3 Technical Reference Architecture

Modular, service-oriented architecture and configurable workflows

The technical reference architecture which is finally derived from the design decisions described in Subsection 3.3.2 and additional requirements implements a concept of a service-oriented architecture, model-based data structures and flows, and generic but customizable UI components.

System functions are divided into encapsulated, exchangeable, and configurable sub-functions. These sub-functions or services can be recombined in many ways to create various workflows which offer different assistance functions. For seamless data exchange between all services, a common component-based metamodel is introduced which is most notably suited for model-based services such as simulation and optimization.

The architecture for operator assistance systems can be divided into three horizontal layers: the data layer, the service layer, and the UI layer, see Figure 3-4. The technical reference architecture provides generic implementations of the core elements in this architecture: the execution engine calling services as specified in workflows, a UI backend, and a data management based on metamodels for component libraries, plants, and workflows.

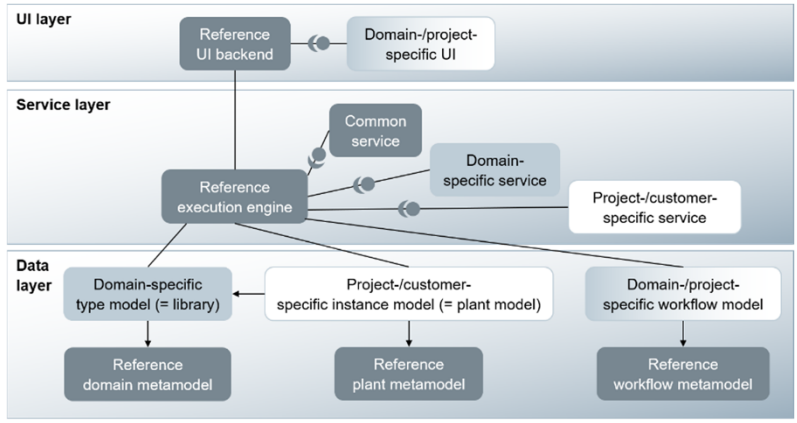


Fig. 3-4: *General technical architecture for an operator assistance system*

Reuse of reference architecture reduces development effort

When implementing a specific operator assistance system, these reference architecture elements form the base. Starting from there, firstly, unspecific or domain-specific frontloading and, finally, project- or customer-specific engineering is performed, see Figure 3-5. Implementing new services or new adapters for existing computational modules such as simulation tools is part of the frontloading. With an increasing number of domains and projects addressed, the reference architecture becomes more elaborate and the collection of reusable services grows. The effort is shifted away from software implementation towards model engineering and system configuration: specifying domain libraries, setting up workflows and data contracts, generating plant models, and configuring UIs. The complete development process is further facilitated by defined process steps, toolkits, and many templates.

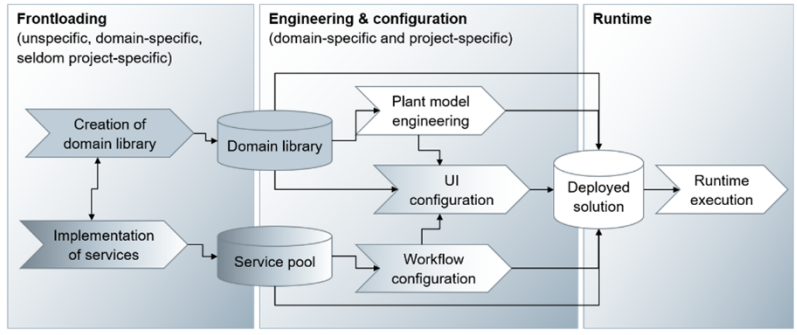


Fig. 3-5: *General steps of the reference-based development process*

www.dbooks.org

3.3.4 Workflow of Services and Data Flow

Collaboration of services via common runtime model

The reference architecture strictly separates the logical and sequential workflow of services and the data flow during runtime execution, as shown for a generic workflow in Figure 3-6. There is no bilateral data exchange between the services. Each service communicates only with the current runtime model and does not know about the source and the destination of any specific variable value. This ensures consistency of data during the whole workflow, simplifies configuration of workflow sequences and data contracts, and guarantees flexibility to replace individual services.

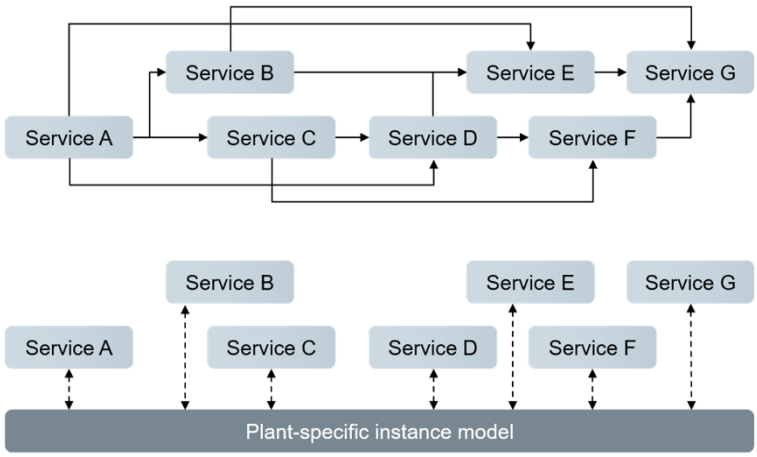


Fig. 3-6: *Workflow (upper part) and data flow (lower part)*

3.3.5 Application Example for an Adaptable and Flexible Factory

Reduced development cost for operator assistance in adaptable and flexible factories

The technical reference architecture presented was implemented as a software framework which was successfully applied in the development of a prototypical assistance system for the operation of an adaptable and flexible factory. The prototype system integrates data from an enterprise resource planning (ERP) system, from a manufacturing execution system (MES), and machine data via the standard communication protocol OPC UA. It contains functions for virtual monitoring of the production, online calibration of the models, detection of any failures and deviations, prediction of critical situations such as bottlenecks, and job shop and flow shop schedule optimization. Figure 3-7 shows the workflows of three of these functions and illustrates how services are reused and re-combined to

offer various functions. Development time was significantly reduced compared to a project- and task-specific development by using the reference architecture as the starting point and core of the system.

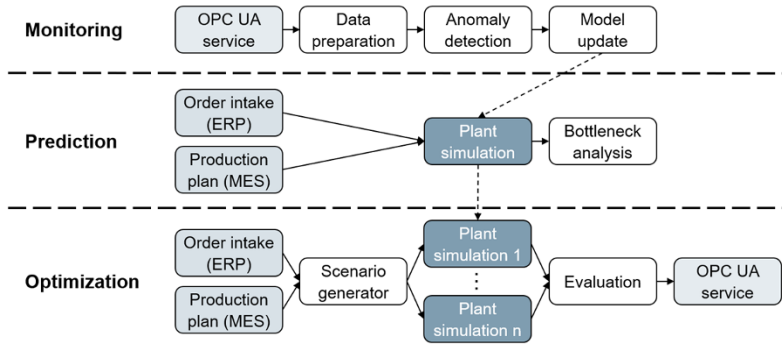


Fig. 3-7: *Workflows for three different assistance functions*

3.4 Checkable Safety Cases for Architecture Design

In this section, we introduce a method for safety argumentation in the design of system and reference architectures. Safety requirements are crucial for CESs and CSGs that may harm people, equipment, or the environment. Adaptable and flexible factories are a typical example of safety-critical systems. Our goal is to support the construction and maintenance of the argumentation that the system architecture of a flexible system satisfies the system safety properties. To this end, we introduce checkable safety cases.

Systems implementing safety functionality that will operate safely in a given operational context must be proven. To this end, more and more safety standards nowadays, such as ISO 26262 [ISO 2018] in the automotive industry, recommend the creation of a safety case. A safety case is a collection of documents entailing an implicit, well-reasoned argument that the system is acceptably safe to operate in a given context, based on certain evidence [Bloomfield and Bishop 2010]. To enable the automated manipulation of safety cases, several approaches for modeling safety cases have been proposed in literature, the most prominent approaches being based either on the Structured Assurance Case Metamodel (SACM) [SACM 2019] or the Goal Structuring Notation (GSN) [GSN 2018].

The validity of the safety case models must be revised every time there is a change in the system specification. However, currently, such validity revision is done manually, implying a considerable amount of effort and costs. Given the frequent changes to architectural

www.dbooks.org

structures of flexible systems, there is a need to automate validity checks for safety cases. To this end, we introduce checkable safety case models with the scope of supporting safety engineers in maintaining valid safety case models given changes in other system models. Checkable safety case models are a special type of safety case model that is integrated with system models, which are amenable to automated checks.

Extension of the SPES_XT modeling framework

To this end, we extend the SPES_XT modeling framework with a new system view, that is, the safety case view. The safety case models are to be integrated with the other system models corresponding to different viewpoints (e.g., requirements viewpoint, logical viewpoint). The safety case model is to be modeled alongside the system development and will be maintained to ensure consistency with other system models during the entire system lifecycle.

To support safety engineers in modeling checkable safety cases, we propose a set of checkable safety case patterns. Similar to design patterns, safety case patterns are templates for re-occurring safety fragments that can be reused in different safety cases [Kelly and McDermid 2010]. These templates entail placeholders for system-specific information which are to be filled when the pattern is used in a certain safety case. We extend the concept of safety case patterns with checkable safety case patterns. Checkable safety case patterns come with a set of automated checks that may be performed on the safety case fragment obtained after the instantiation of the pattern.

Modeling checkable safety case fragments for reference architectures

Among other things, the safety case of a system must entail an argument about the satisfaction of safety properties by the system architecture. As reference architectures are blueprints to be used for modeling system architectures, for each such reference architecture we provide a pattern for arguing about the fact that the reference architecture satisfies certain safety properties. When the architecture of a certain system uses a certain reference architecture as a blueprint, the corresponding safety case checkable pattern can be used to model the safety argumentation for the constructed system architecture.

3.4.1 Checkable Safety Case Models – A Definition

Safety case models on which automated checks can be executed

To support safety engineers in the cumbersome, time-consuming process of keeping safety case models consistent with system models (e.g., system architecture models), we propose checkable safety cases.

The validity of checkable safety case models is checked by the automatic execution of sanity checks, based on explicit specification of semantics of safety case elements, and the integration of the safety case model with system models and automated verification approaches [Cârlan et al. 2019], see Figure 3-8.

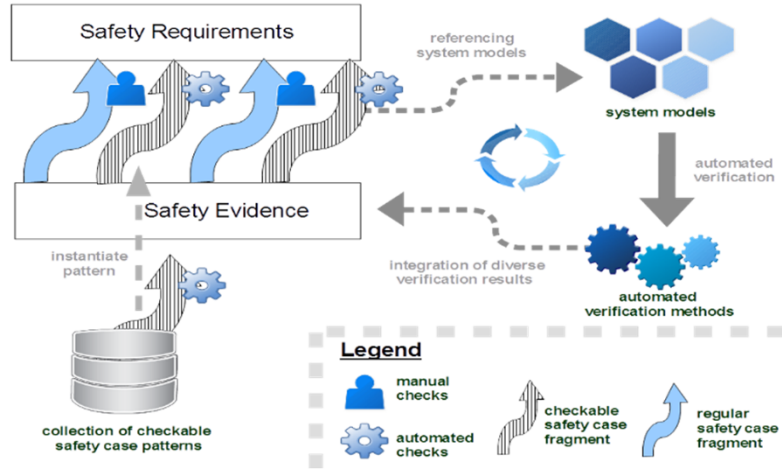


Fig. 3-8: *Safety argumentation based on contract-based verification*

Given a change in a system model that is traced from the safety case model, consistency checks between the safety case model and the system models are automatically executed. These consistency checks assess whether the argumentation is still valid considering the changes in the system model that the argumentation applies to. Then, the safety engineer must update the safety argument in accordance with the changes, while also generating the evidence required. Given that system models are amenable to automated checks, the results of such checks can be used as evidence in safety cases. Therefore, we integrate safety case models with such automated verification approaches, thus enabling 1) automatic detection of stale evidence, and 2) automatic integration of new verification results as evidence, while assessing the impact of the new evidence on the confidence in the overall argumentation.

Checkable safety cases entail checkable and non-checkable argumentation fragments

Checkable safety case models entail both checkable and non-checkable argumentation fragments that are connected with each other. On the one hand, non-checkable argumentation fragments entail regular safety case elements, as defined by the Goal Structuring Notation (GSN) — a standardized graphical notation for describing safety cases and currently the most frequently used language for

www.dbooks.org

modeling safety cases [GSN 2018]. On the other hand, checkable safety case fragments entail a set of interconnected specialized safety case elements. Specialized safety case elements extend GSN, with each specialized element representing a reoccurring claim in safety cases, thus having certain semantics. Specialized safety case elements reference certain types of system model elements or entail metadata regarding certain verification approaches. They may be connected to each other only via specialized connections, which extend the connections specified in GSN. In contrast to GSN-based connection types that ensure the correct construction of arguments from a semantic point of view, specialized connections enable intrinsic checks on safety case models, which ensure the construction of semantically correct arguments.

3.4.2 Checkable Safety Case Patterns

To support safety engineers in modeling checkable safety cases, we propose an exemplary set of checkable safety case patterns.

Checkable safety case patterns enhance state-of-the-art patterns to enable automated checks

While the argumentation structure of checkable safety case patterns is based on state-of-the-art patterns, the connected elements the structure contains are specializations of regular safety case elements. The specialized safety case elements have variable declarations, which are placeholders for a reference to a certain type of system element or verification information. The variables are to be instantiated with specific references when the pattern is used to model the safety case of a certain system. The relationships among specialized safety case elements are described via dedicated connections, thus enabling intrinsic consistency checks, which prohibit pattern misuse — a specialized safety case element may be connected only to certain types of other specialized safety case elements.

A checkable safety case pattern is specified as presented in the following [Kelly and McDermid 2010]. We extend the specification of regular safety case patterns with information specific to checkable safety case patterns:

- *Name*: the identifying label of the pattern giving the key principle of its argument
- *Intent*: the goal the pattern is trying to achieve
- *Motivation*: the reasons that gave rise to the pattern and the associated checks
- *Structure*: the structure of the argument in GSN

- *Participants*: each element in the pattern and its description; here we differentiate between plain SACM-based elements and specialized elements — for the specialized elements, the corresponding metadata is explained
- *Collaborations*: how the interactions of the pattern elements achieve the desired effect of the pattern; here we explain the specialized connections among the specialized elements and how the specialized safety case elements will be connected with the regular elements
- *Applicability*: the circumstances under which the pattern could be applied, that is, the necessary context
- *Consequences*: what remains to be completed after pattern application
- *Implementation*: how the pattern should be applied; here we discuss how the safety case elements are to be instantiated

The following documentation information is specific to checkable safety case patterns:

- *Prerequisites*: regarding the existence of certain system models or of certain verification tools
- *Automated checks*: the checks that can be executed on the safety case fragments produced after the instantiation of the pattern

3.4.3 An Example of Checkable Safety Case Patterns

Arguing about the satisfaction of a certain safety property by an architecture

In Figure 3-9, we present part of the checkable safety case fragment concerning the satisfaction of a certain safety property by a system architecture built in a contract-based manner. The system architecture entails assume-guarantee (A/G) contracts that formalize safety properties. The properties are satisfied if: 1) the contracts of the architecture model are correctly refined by the contracts of the components within the architecture model (claim expressed as *Refinement Check* specialized goals); 2) the contracts of the architecture components are satisfied (claim expressed as *Compatibility Check* specialized goals); and 3) each architecture component correctly implements its contracts (claim expressed as *Implementation Check* specialized goals). Each claim in the argument is a specialized safety case element, with a certain meaning and with certain references to system model elements. Given specialized connections between specialized elements, intrinsic consistency checks are enabled. For example, elements of the type *CBD Strategy* may be supported only by goals of the type *Compatibility Check*,

www.dbooks.org

Refinement Check, and *Implementation Check*, ensuring the validity of the argument structure. The *CBD Strategy* references a certain component in the system architecture that will implement the safety contract. Consequently, to ensure the validity of the argumentation, we check whether the sub-goals of the type *Implementation Check* supporting *CBD Strategy* reference only children of the component referenced by the strategy. The validity of claims of the type *Implementation Check* is checked via an automated verification tool able to check architecture models annotated with contracts — a model checker. In the example presented in Figure 3-9 the model checker used is NuSMV [Cimatti et al. 2002].

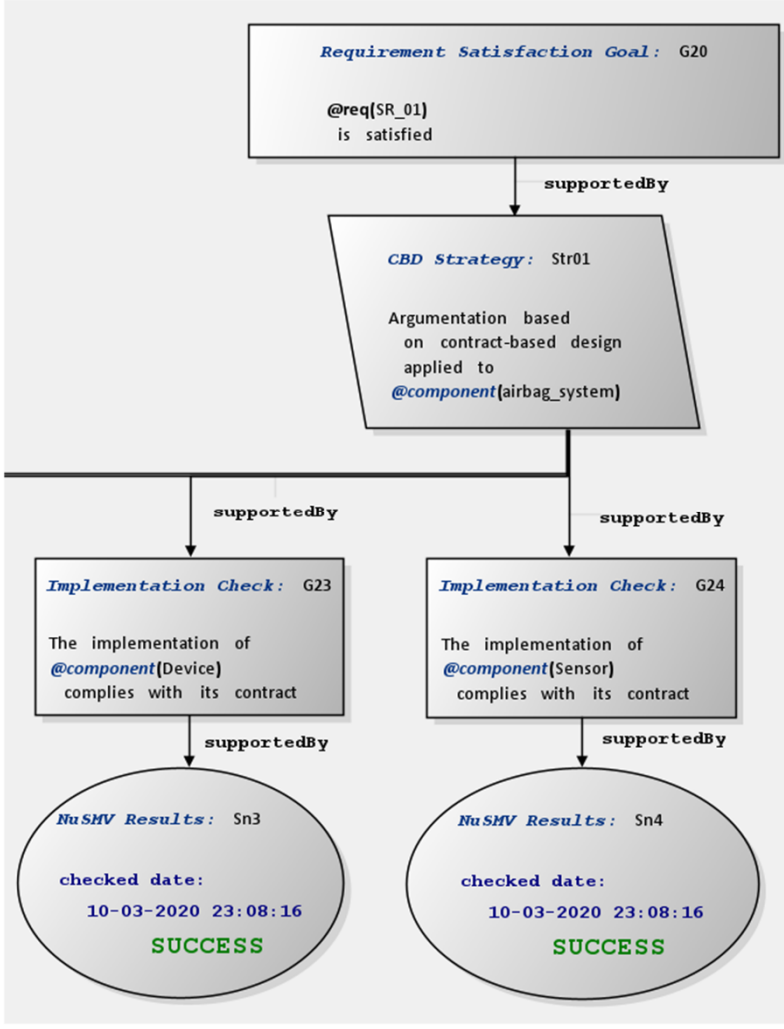


Fig. 3-9: *GSN-based safety case fragment*

In Figure 3-9 a GSN-based safety case fragment is shown arguing about the verification via NuSMV model checker of the system architecture model against system safety properties specified as contracts. Due to space constraints, the figure displays only part of the argumentation, namely the argumentation legs regarding the correct implementation of the subcomponents of the architecture.

In conclusion, we propose the creation of checkable safety case patterns that argue about the implementation of safety properties by a system architecture which may also be based on a certain reference architecture. Given the specialized safety case elements contained in the pattern and their integration with system models and verification tools, the validity of the argumentation fragment resulting from the pattern instantiation is automatically checked if there is a change in the corresponding system architecture model. These automated checks are especially needed if there are frequent changes.

3.5 Conclusion

In this chapter, we presented a general method for designing reference architectures and deriving system architectures for CESs and their CSGs in order to support reuse of system architectures. In addition, the method can be used to design a CSG and the interfaces of collaborating CESs within this CSG. In a next step, the architectures of the CES can be refined based on the reference architecture. This enables the integration of CESs of different organizations within one CSG. As an application example, we provided a short overview of the reference architecture for adaptable and flexible factories, detailed by a CES implementing an operator assistance system. The technical reference architecture for this CES shows the reuse potential for various operator assistance systems and provides a promising basis for future systems.

In order to consider non-functional requirements in the system architecture, we also introduced checkable safety case models. These checkable safety cases support maintenance of the validity of safety case models and keep them consistent with system architecture. This method may be used for the construction of the safety argumentation system architectures based on reference architectures.

In addition to the methods presented, we also developed prototypical tools which support and facilitate the application of the methods. The methods and reference architectures presented in this chapter have been applied successfully but should nevertheless be

www.dbooks.org

applied to other CESs and their CSGs to prove their benefits. Future research may extend them beyond their current scope, for example, by involving artificial intelligence as design support as well as considering artificially intelligent CESs and CSGs in particular.

3.6 Literature

[Bloomfield and Bishop 2010] R. Bloomfield, P. Bishop: Safety and Assurance Cases: Past, Present and Possible Future – an Adelard Perspective. In: *Making Systems Safer,* Springer, London, 2010, pp. 51-67.

[BMWi 2017a] BMWi: Platform Industrie 4.0 – Aspects of the Research Roadmap. In Application Scenarios. https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf; accessed on 07/07/2020.

[BMWi 2017b] BMWi: Platform Industrie 4.0 – Fortschreibung der Anwendungsszenarien der Plattform Industrie 4.0. https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html; accessed on 07/07/2020 (available in German only).

[Böhm et al. 2018] B. Böhm, M. Zeller, J. Vollmar, S. Weiß, K. Höfig, V. Malik, S. Unverdorben, C. Hildebrandt: Challenges in the Engineering of Adaptable and Flexible Industrial Factories. In: I. Schaefer, L. Cleophas, M. Felderer (eds.): Workshops at Modellierung 2018, Modellierung 2018, Braunschweig, Germany, February 21–23, 2018, pp. 101–110.

[Böhm et al. 2020] B. Böhm, J. Vollmar, S. Unverdorben, A. Calà, S. Wolf: Holistic Model-Based Design of System Architectures for Industrial Plants. In: VDI – Verein Deutscher Ingenieure e.V. (eds.): Automation 2020, Baden-Baden, 2020.

[Boschert et al. 2018] S. Boschert, R. Rosen, C. Heinrich: Next Generation Digital Twin. In: Proceedings of the 12th International Symposium on Tools and Methods of Competitive Engineering — TMCE 2018, Delft, 2018, pp. 209-218.

[Cârlan et al. 2019] C. Cârlan, V. Nigam, S. Voss, A. Tsalidis: ExplicitCase: Tool-Support for Creating and Maintaining Assurance Arguments Integrated with System Models. In: Proceedings of *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* 2019, pp. 330-337.

[Cimatti et al. 2002] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: *Computer Aided Verification. CAV 2002.* Lecture Notes in Computer Science, vol 2404. Springer, Berlin, Heidelberg.

[de La Vara et al. 2016] J. L. de La Vara, M. Borg, K. Wnuk, L. Moonen: An Industrial Survey of Safety Evidence Change Impact Analysis Practice. In: *IEEE Transactions on Software Engineering*, *42*(12), 2016, pp: 1095-1117.

[GSN 2018] Assurance Case Working Group. Goal Structuring Notation Community Standard (Version 2). https://www.goalstructuringnotation.info/; accessed on 01/11/2020.

[ISO/IEC/IEEE 42010 2011] International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers (eds.): Systems and Software Engineering - Architecture Description.

Geneva, s.l., New York, http://ieeexplore.ieee.org/servlet/opac?punumber=6129465; accessed on 04/04/2020.

[ISO 2018] International Organization for Standardization (ISO): 26262: Road Vehicles - Functional Safety.

[Kelly and McDermid 2010] T. Kelly and J. McDermid, "Safety case patterns-reusing successful arguments," IEE Colloquium on Understanding Patterns and Their Application to Systems Engineering (Digest No. 1998/308), London, UK, 1998, pp. 3/1-3/9.

[Lin et al. 2017] S.-W. Lin, M. Crawford, S. Mellor (Eds.): The Industrial Internet of Things Volume G1: Reference Architecture. Industrial Internet Consortium (IIC) Technology Working Group, 2017. http://www.iiconsortium.org/IIC_PUB_G1_V1.80_2017-01-31.pdf; accessed on 07/07/2020.

[Rosen et al. 2018] R. Rosen, S. Boschert, A. Sohr: Next Generation Digital Twin. In: atp magazin, Atp-Mag. 60, 2018, pp. 86–96.

[Rosen et al. 2019] R. Rosen, J. Jaekel, M. Barth, O. Stern, R. Schmidt-Vollus, T. Heinzerling, P. Hoffmann, C. Richter, P. Puntel Schmidt, C. Scheifele: Simulation und Digitaler Zwilling im Engineering und Betrieb automatisierter Anlagen - Standpunkte und Thesen des GMA FA 6.11. In: VDI – Verein Deutscher Ingenieure e.V. (eds.): Automation 2019, Baden-Baden, 2019 (available in German only).

[SACM 2019] Structured Assurance Case Metamodel. URL https://www.omg.org/spec/SACM/About-SACM/; accessed on 04/11/2020.

[Unverdorben et al. 2019] S. Unverdorben, B. Böhm, A. Lüder: Concept for Deriving System Architectures from Reference Architectures. In: 2019 IEEE International Conference on Industrial Engineering and Engineering Management: IEEM2019: Dec. 15-18, Macau/IEEE International Conference on Industrial Engineering and Engineering Management - [Piscataway, NJ]: IEEE, 2019, pp. 19-23.

[VDI/VDE 3695 2010] Association of German Engineers (VDI), Association for Electrical, Electronic & Information Technologies (VDE): VDI 3695 Blatt 3 - Engineering of Industrial Plants - Evaluation and Optimization - Subject Methods. 2010.

[Zhou et al. 2019] Y. Zhou, T. Schenk, M. Allmaras, A. Massalimova, A. Sohr, J. C. Wehrstedt: Flexible Architecture to Integrate Simulation in Run-Time Environment. Presented at the Automation Congress 2019, VDI, Baden-Baden, 2019.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Alexander Hayward, Helmut Schmidt University Hamburg
Marian Daun, University of Duisburg-Essen
Ana Petrovska, Technical University of Munich
Wolfgang Böhm, Technical University of Munich
Lisa Krajinski, University of Duisburg-Essen
Alexander Fay, Helmut Schmidt University Hamburg

4

Function Modeling for Collaborative Embedded Systems

The evolution from traditional embedded systems to dynamically interacting, collaborative embedded systems increases the complexity and the number of requirements involved in the model-based development process. In this chapter, we present the new aspects that need to be considered when modeling functions for collaborative embedded systems and collaborative system groups, such as the relationship between functions of a single system and functions resulting from the interplay of multiple systems. These different aspects are represented by a formal, domain-independent metamodel. To aid understanding, we also apply the metamodel to two different use cases.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_4

4.1 Introduction

In modern development methodologies for complex systems, the modeling of functions represents a historically grown and proven way of dealing with large quantities of requirements that need to be taken into account. A function can be used to describe the purpose of a system at different levels of detail.

The SPES projects

The SPES2020 and SPES_XT projects (cf. [Pohl et al. 2012, Pohl et al. 2016]) have already developed a comprehensive set of science-based methods for modeling and analyzing functions of embedded systems, with a special focus on consistency and semantic coherence as part of a comprehensive methodological framework. The methods are based on the assumption that the embedded systems under development are to be integrated into a static context that is well known at the time of development.

Goal fulfillment via collaboration

The additional assumption considered in CrESt—that individual systems no longer achieve the goals[1] associated with them alone, but rather by collaboration with other systems—results in a range of new challenges for which the existing SPES modeling framework is no longer sufficient and needs to be extended.

CES and CSG

A collaborative embedded system (CES), collaborating with other CESs that may be instances of different system types, should be able to achieve goals that 1) the CES could not achieve alone, or 2) could be achieved more easily or better by combining their functions with other CESs. For collaboration, the participating CESs form a common group, referred to as a collaborative system group (CSG). Since a CSG constitutes itself dynamically at runtime and its members, goals, and functions can change, methods for mastering the complexity are particularly necessary for modeling functions at CSG level.

Outline

In this chapter, we describe the new aspects that have to be considered when modeling functions for both CESs as well as the resulting CSG. To describe these aspects systematically, we use a metamodel. With regard to the derivation of this metamodel, Section 4.2 describes the requirements and aspects on which it is based. In Section 4.3, we provide further background information. We then present the metamodel on a domain-independent level in Section 4.4 and evaluate it in Section 4.5. To enable a better understanding of the metamodel, we apply it to two use cases in Section 4.6, and this is

[1] See Chapter 2 for a detailed discussion

www.dbooks.org

followed by related work in Section 4.7 and the conclusion in Section 4.8.

4.2 Methodological Approach

Deriving requirements

Model-based continuous function-centered engineering processes are already established in engineering practice [Daun et al. 2019a]. To support function-centered development, this chapter proposes a function metamodel. In order to achieve the goal of defining a uniform modeling methodology for functions, we have applied the following research methodology. To ensure applicability to a variety of domains and in various contexts, first, we have gathered requirements from academia and industry, using common requirements elicitation techniques such as interviews, workshops, and in-depth discussions. In addition, we have investigated engineering methods of different domains to foster function-centered engineering. As a result, we have derived a set of seven high-level requirements. For details of the requirements elicitation phase and the results, please refer to [Ludewig et al. 2018]. The following is a brief outline of the major requirements:

Requirements 4-1*: Requirements for the function metamodel*

- **Req.1:** It must be possible to model functions on different abstraction layers. Composition and decomposition of functions must consider the relationship between single functions of CESs and overall functions on the CSG layer. On the highest layer of abstraction, a function can be understood as the emerging result of individual contributing functions. On the lowest abstraction layer, it must be possible to model an atomic function and its contribution to the overall function.
- **Req.2:** It must be possible to model which overall function of a CSG an individual function of a CES can contribute to and how it can do so. For this purpose, the modeling process must consider whether different inputs and outputs of functions can be connected with those of other functions with regard to compatibility.
- **Req.3:** Due to characteristics of openness and dynamicity, functions—as well as their connections to each other—may vary or change over time. These possible changes in individual functions may affect the overall function. Therefore, the possible inputs and outputs of functions—on different abstraction layers—that vary over time must be considered in the modeling process.

- **Req.4:** Functions transform input into output to achieve a goal or meet a requirement. It must be possible to model the relationship between a goal (or a requirement) and its possible solution provided by a function. This modeling must consider priorities of goals and functions as well as conflicts between them. This modeling must also include dependencies between different functions.
- **Req.5:** In case of a failure or of an error in individual functions, compensation strategies are necessary. Possible functional errors or failures must be considered in modeling to make them detectable. Therefore, relationships between the function model and other system models must be considered.
- **Req.6:** Since different CSG functions are realized by different systems, potentially from different manufacturers, modeling approaches must ensure that inputs and outputs of different functions are compatible and suitable/consistent with each other.
- **Req.7:** CESs must provide the functionality to allow the CSG to be restructured at runtime. This leads to functions being related to different states. These states must represent, for example, when a function is accessible and when the transformation from input to output is not available.

Metamodel development

We subsequently developed a metamodel iteratively to satisfy these requirements. We conducted workshops with the stakeholders to negotiate and re-iterate the metamodel as long as necessary to achieve a final, agreed version that fits all purposes for functional modeling and analysis of CESs.

The following is an example of the analysis of the individual methods investigated and the resulting rationale for the relevance of the method for the function metamodel. For demonstration purposes, we use a method for modeling the goals of CESs and CSGs.

Example 4-2: Functional aspects of the goal modeling method

Method name: GoalBasedSystemGroupEngineering

Metamodel extension required: Consider the relationship between goals and functions.

Reason: This method focusses on the definition of goals for the individual CESs as well as for the CSG. In the relationship between the CSG goals and the CES goals, it must be ensured that every CSG goal can be operationalized. Therefore, goals are refined into tasks, which represent abstract definitions of functions to be implemented.

www.dbooks.org

4.3 Background

SPES modeling framework

Our work builds on results from the SPES projects that provide a framework that enables seamless model-based engineering of embedded systems (cf. [Pohl et al. 2016]). The SPES modeling framework includes a *functional viewpoint* for specifying the system's functionality. The system's functionality is elicited from the requirements in a preceding requirements viewpoint. Our metamodel builds on this background work.

FOCUS theory

To a large extent, the SPES modeling framework is based on a formal theory called FOCUS, which provides models and formalisms for specifications and the development of distributed interactive and dynamic systems. FOCUS establishes a formal semantics that serves as a common ground for also giving means to functional behavior. In FOCUS (cf. [Broy and Stølen 2012], and [Broy 2014]), a system's interface is determined by the system's boundary. The *syntactic interface* describes the set of input and output channels and the message types that these channels transport across the system's boundary. The system's functionality is described by the *interface behavior,* which can be observed at the system's boundary and which is defined by the history of streams of messages across the input and output channels. The histories of the streams of messages across the input and output channels capture the system's interface behavior. Accordingly, the interface behavior models system functionality that can be observed.

4.4 Metamodel for Functions of CESs and CSGs

Five major aspects

The metamodel for functions is given in Figure 4-3, which shows the aspects to be considered when modeling functions of CESs and CSGs. Based on the requirements, we have identified five major aspects that need to be considered when modeling functions for CESs and CSGs. These aspects are detailed in the subsections below:

- First, a differentiation between individual collaborating systems, not collaborating systems, and collaborative system groups is necessary. The CESs can partake in a CSG to fulfill their purposes as well as to contribute to the goals of the CSG. Whether a function belongs to either a CES or the CSG influences whether that function exists on its own or only in the collaborative interplay.

- Second, the term function must be defined, thereby placing a particular emphasis on its behavior and its interfaces. It is important to identify how these aspects relate to the function's contribution to the collaboration of the CSG.
- Third, as collaborative systems are inherently goal-oriented, system goals must be considered. System goals represent the established realization of stakeholder goals, which are elicited during requirements engineering. This means that a CES takes part in a collaboration only if this fulfills a certain purpose (i.e., the system goal), depending on the needs of the CSG and the other CESs. The systems offer different functions to optimize goal fulfillment of the individual CESs as well as the overall CSG.
- Fourth, roles play a vital part in the engineering of collaborative systems, as the functions a system offers and requires depend on the role the system takes in a certain collaboration.
- Fifth, context and adaptivity must be considered. CESs and CSGs operate in an open, dynamic context. The dynamicity of the operational context is the main trigger for the adaptation of the entire CSG, which might result in reconfiguration of the individual CESs and thus impact their functional interplay.

4.4.1 Systems, CESs, and CSGs

Collaborating and not collaborating systems

In this section, we introduce the relationship between a system and the collaboration of systems. Therefore, we start—as in traditional system analysis—by separating a *system* and its *context*. For a system, we must distinguish whether it is a *not collaborating system*, a *collaborating system,* or a *CSG*, always viewed at a certain point in time.

Belonging to a certain CSG depends on time

A not collaborating system does not collaborate with other systems in a given CSG at a current time *t*. We can distinguish between *CESs* and *non-CESs*. While a non-CES cannot collaborate in a CSG at any time[2], a CES can become a collaborating system for a CSG at a later point in time during the runtime of the system. A collaborating system is part of a CSG, which consists of multiple collaborating systems. Within a CSG, the CESs work together, provide their functions to each other, and share information to promote common CSG goals.

[2] Note: CESs and non-CESs are always related to a specific type of CSG: a system can be collaborating with respect to a given type of CSG and a non-CES for another type of CSG.

www.dbooks.org

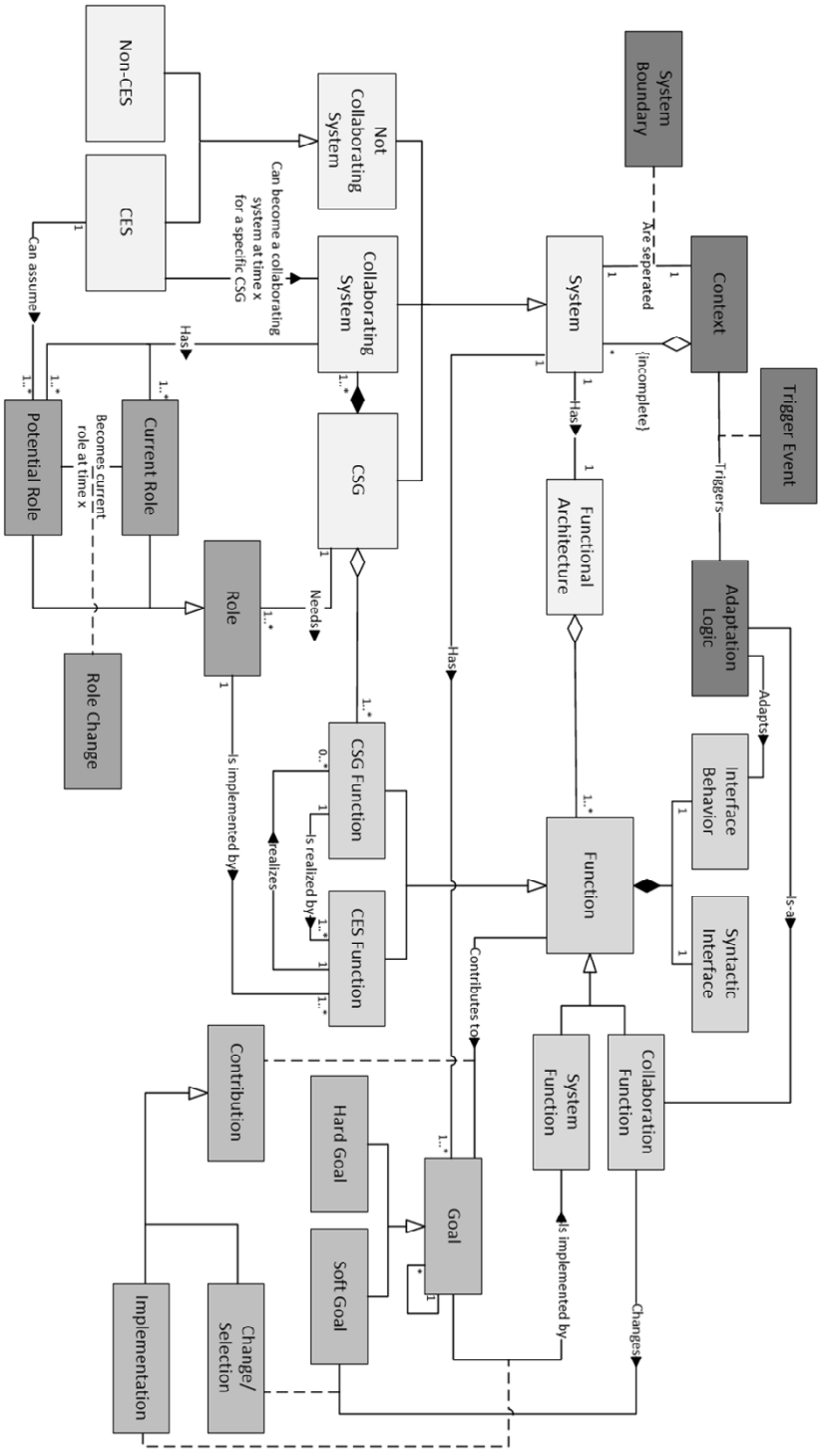


Fig. 4-3: *Metamodel for functions of CESs and CSGs [Hayward et al. 2020]*

Functional architecture

Every system (i.e., each CES, each CSG, each non-CES) has a *functional architecture* that contains all the functions of the system and describes how the functions interact with each other to achieve *goals*. This means that each individual CES consists of a functional architecture (cf. [Pohl et al. 2016]) which is therefore part of the larger functional architecture of the CSG.

4.4.2 Functions

Functions have interfaces

Systems can be described on different levels of detail by their *functions* [VDI 2221]. Functions describe the behavior of a system through the interrelation between input and output variables [VDI 2222]. A function has a *syntactic interface*, through which it can take up information, material, or energy, transform it and output it again. Depending on the domain, the understanding of the term function can vary slightly in detail but this definition is valid at this general level [Eisenbart et al. 2012].

Functional decomposition

In the classical design methodology according to [Pahl et al. 2013] as well as in today's model-based system development [Vogelsang 2015], [Meissner et al. 2014], functions are derived from requirements lists and models during an early design phase to capture the required functionality of the CES. Since specific solution principles can be derived only to a very limited extent based on these abstract functions, the functions are further decomposed into sub-functions, which can also be further subdivided, thus forming a hierarchy. These sub-functions again have interfaces through which they are connected. The functional hierarchy is called *functional architecture*. The functions and the resulting functional architecture can be used to describe what a system should be able to do. Additionally, the *interface behavior* can be used to describe which states, state transitions, and functional dependencies functions have [Eisenbart et. al. 2012].

Functions and collaboration

A CES no longer performs certain functions alone; it performs them in collaboration with other CESs. For this purpose, the CESs form a CSG and thereby provide their functions (*CES function*) to each other to achieve a common goal. The CSG can be considered as a system on its own again with components and functions (*CSG function*). These functions of the CSG are realized by CES functions and result from the interplay between the collaborating systems. While the functions of individual CESs can be modeled and realized during design time, the functions of the CSG are only constituted through collaboration between several CESs at runtime. By modeling CSG functions, we can

www.dbooks.org

indicate the specific contributions involved that CESs have to provide to achieve common goals of the CSG.

System function and collaboration function

In addition to these two specializations of the function concept, the metamodel further distinguishes between *system function* and *collaboration function*. A system function contains the individual contribution of a CES within the collaboration. In a broader sense, a system function also includes other internal functions of the CES that contribute, albeit indirectly, to the fulfillment of goals. Collaboration functions, on the other hand, comprise a set of functions that are assumed to be available in all CESs participating in the CSG to enable collaboration in general and independently of the specific form and goal of the CSG. These collaboration functions include functions for the perception of and communication with other CESs, the negotiation of goals, the comparison of required and existing functions, or the adaptation of the CESs' behavior to meet the conditions of the CSG.

4.4.3 Goal Contribution and Fulfillment

Goals

Systems have *goals* associated with them. A goal is thus defined as a condition or situation the system wants to achieve or a behavior the system wants to exhibit. This holds for an individual CES as well as for a CSG. As mentioned in the previous sections, the fulfillment of these goals is always realized through functions and their implementation with the help of algorithms. This explicit manifestation of the stakeholder goals enables systems to fulfill the goals planned at the time of development during operation. We have to consider several situations according to system goals.

Negotiating goals

The goals of the CESs concerned may differ from each other and from the goals of the CSG and may even be conflicting goals. In the case of (partly) different or contradictory goals, and in order to form a functioning CSG, the following must be negotiated: different goals between CESs collaborating in the CSG, differences between CES and CSG goals, and the way each CES contributes to the achievement of the CSG goals. Finally, the individual goals must be adopted in order to reach a consistent goal system within the CSG and its collaborating systems. As a consequence, it must be possible for the CESs to change their goals according to the results of the negotiations. In this sense, goals are considered dynamic at runtime.

Hard goals and soft goals

We differentiate between goals that will never be changed (*hard goals*) and goals that may be changed (s*oft goals*) in order for a CES to

actively contribute to the CSG goal system[3] in a consistent manner. To this end, we have extended the function metamodel from SPES to match these additional requirements.

4.4.4 Roles

Roles enable assignment of responsibilities

Another concept that supports modeling and implementation of CSGs is that of *roles*. Within a CSG, different functions are needed to achieve the CSG goals. Roles can be used to define, within the CSG, which collaborating system is responsible for which CES functions and thus for which goals. CESs assume roles when they join the CSG. A single CES can potentially assume one or more roles within a CSG at the same time. The roles allow the definition of the necessary CES functions and thus the necessary behavior of the individual collaborating systems [Weiß et al. 2019, Regnat et al. 2019].

Potential roles and current role

A CES that has assumed a certain role within the CSG (*current role*) is responsible for the role-related functions. If a CES leaves a CSG (e.g., intentionally or due to an error), but the functions associated with its role must still be provided, it may be necessary for another CES, which has the necessary functions, to change its current role. This *role change* is only possible if the functions of this subsequent CES allow (*potential role*), it to assume the role from the leaving CES. These processes are only possible if the CESs involved in the CSG have a common understanding of the roles to be assumed.

4.4.5 Context and Adaptivity

System boundary

A system is separated from its context and other systems by its *system boundary*. The system boundary defines whether an object belongs to the system or is outside of it. However, because of the dynamicity of the CSG, the boundary, the behavior, and the structure of the CSG may change over time. Consequently, we had to extend the metamodel to cope with such situations — namely, a potential CES that is outside the CSG at a given point in time may enter the CSG and therefore become part of the CSG structure (i.e., will be inside the CSG boundary).

Context

The *context* of the system describes the environmental surrounding that is not part of the system. The surrounding includes

[3] Note: The terms hard goal and soft goal are used differently here as compared to goal modeling literature: hard goals are not subject to negotiation and are in a sense "static," while soft goals may be dynamically negotiated and changed in order to cooperate in a CSG.

www.dbooks.org

persons, groups, organizations, processes, events, documents, functions of other systems, etc. In other words, the context is a perceivable part of the environment that consists of all the objects relevant to the system. Context is everything that is relevant to a system but remains external to its range of action [Lalanda et al. 2013]. Separating the system and its context means distinguishing between changeable and unchangeable variables [Pohl 2010]. Consequently, the context consists of all objects relevant to the system but outside the system's boundary.

Context and system boundaries change

Modern systems, such as CESs and CSGs, operate in a changing, uncertain, and dynamic context. In addition to the dynamicity of the context in which the systems operate, the structure of the system itself is also dynamic. Consequently, as briefly explained earlier in this section and in Section 4.4.1, the structure and the behavior of the system fluctuate over time — namely, a CES that is not collaborating at time x might become a collaborating system at a definite time $t + x$ in the future by joining a specific CSG; and vice versa, a collaborating system of a CSG might, over time, leave the group. This directly impacts and changes the boundary of a system, which is no longer static, changes at runtime, and goes beyond what was defined during the system's design.

Trigger events

Consequently, the systems must be able to adapt in order to deal with the dynamicity and the runtime changes that might originate externally from the context in which the system operates, as well as internally from the system itself. We refer to these uncertainties and changes that trigger the adaptivity as *trigger events*. What we distinguish as internal or external events depends specifically on whether the system under consideration is a CES or a CSG. From the point of view of an entire CSG, internal trigger events could refer to, for example, the changes in CESs that collaborate, as role changes. In contrast, an external trigger event could be a CES from the context requesting to join and share its functionality with the CSG. However, from the view of an individual CES, the changing of a role, which is an internal trigger from the perspective of the CSG, can be considered an external trigger for the CES. In contrast, an example of internal triggers for the CESs are sensor uncertainties, such as sensor ambiguity, sensor imprecision, or even complete sensor failure, which could potentially lead the complete CES to a non-deterministic or faulty behavior.

Adaptivity

In a nutshell, the general idea behind adaptivity is the ability to change the system's observable behavior, structure, and realization [Broy 2017], [Krupitzer et al. 2015], [Giese et al 2013] as a response

to the internal and external events in order for the systems to continue meeting their functional specifications while preserving the performance (or another quality objective) — despite all the changes that the system may encounter during runtime [Petrovska and Pretschner 2019].

Adaptation logic

The adaptivity is enabled by the *adaptation logic*, which is a necessary precondition for a system to adapt to these changing situations. If the adaptation is triggered manually by an external user or administrator (a human assumes the role of an adaptation logic), then this is referred to as a system reconfiguration. In contrast, if the adaptation is triggered and executed by the system itself, in an automated manner, without any user interaction, then we call it self-adaptation [Petrovska et al. 2020]. Specifically, in our metamodel, we consider the adaptation logic to be a collaboration function which adapts the functions and the behavior of the CESs and the CSG through the collaboration of the systems.

4.5 Evaluation of the Metamodel

In this section, we will briefly outline how the proposed function metamodel fulfills the requirements from Section 4.2. Further evaluation is subsequently given in Section 4.6 by showing the applicability of the proposed function metamodel.

4.5.1 Abstraction

Requirements 4-4*: Req.1*

It must be possible to model functions on different abstraction layers. Composition and decomposition of functions must consider the relationship between single functions of CESs and overall functions on the CSG layer. On the highest layer of abstraction, a function can be understood as the emerging result of individual contributing functions. On the lowest abstraction layer, it must be possible to model an atomic function and its contribution to the overall function.

This requirement is fulfilled because a *function* is composed of other functions, thereby allowing the description of functionality at different levels of granularity. Furthermore, the separation between *CSG function* and *CES function* introduces another abstraction layer, as a function belongs either primarily to the overall CSG or to an individual CES. However, in both cases, the functions must be

www.dbooks.org

implemented in a CES, as the CSG relies on the CESs for any kind of resource. In addition, the distinction between *collaboration function* and *system function* also indicates different levels of granularity to describe functional properties.

4.5.2 Relationships between Functions

Requirements 4-5*: Req.2*

It must be possible to model which overall function of a CSG an individual function of a CES can contribute to and how it can do so. For this purpose, the modeling process must consider whether different inputs and outputs of functions can be connected with those of other functions with regard to compatibility.

The aforementioned differentiation between *CSG function* and *CES function* allows us to define which CSG function *is realized by* which CES functions, and which CES function *realizes* which CSG functions.

4.5.3 Openness and Dynamicity

Requirements 4-6*: Req.3*

Due to characteristics of openness and dynamicity, functions—as well as their connections to each other—may vary or change over time. These possible changes in individual functions may affect the overall function. Therefore, the possible inputs and outputs of functions—on different abstraction layers—that vary over time must be considered in the modeling process.

To address this requirement, the *adaptation logic* reacts to t*rigger events* in the *context* and *adapts* the *behavior* of a *function*. A change in an individual function also affects other functions of the CES or the CSG (see Sections 4.5.1 and 4.5.2). In particular, the composition of the *functional architecture* of any kind of *system* may be changed.

4.5.4 Goal Contributions

Requirements 4-7*: Req.4*

Functions transform input into output to achieve a goal or meet a requirement. It must be possible to model the relationship between a goal (or a requirement) and its possible solution provided by a function. This modeling must consider priorities of goals and functions as well as conflicts between them. This modeling must also include dependencies between different functions.

A *goal* is defined as either a *hard goal* or a *soft goal*; each can be decomposed and be related to each other. Each *goal is implemented by* at least one *system function*, while any *function* can contribute to any goal. In addition, a *collaboration function* may *change* a *soft goal*.

4.5.5 Relationships Between Functions and Systems

Requirements 4-8*: Req.5*

In case of a failure or of an error in individual functions, compensation strategies are necessary. Possible functional errors or failures must be considered in modeling to make them detectable. Therefore, relationships between the function model and other system models must be considered.

As mentioned earlier, *functions* and *systems* can be directly related by means of the *functional architecture*. Furthermore, the metamodel differentiates between the *CSG* and *CSG functions*, and between *CESs* and *CES functions*.

4.5.6 Input/Output Compatibility

Requirements 4-9*: Req.6*

Since different CSG functions are realized by different systems, potentially from different manufacturers, modeling approaches must ensure that inputs and outputs of different functions are compatible and suitable/consistent with each other.

Each *function* is defined by its *behavior* and its *interface*. This allows us to check the compatibility of functions. Furthermore, as outlined above, sophisticated relationships between functions, systems, CES functions, and CSG functions can be defined.

www.dbooks.org

4.5.7 Runtime Restructuring

Requirements 4-10*: Req.7*

CESs must provide the functionality to allow the CSG to be restructured at runtime. This leads to functions being related to different states. These states must represent, for example, when a function is accessible and when the transformation from input to output is not available.

The *adaptation logic* allows restructuring functions and functional architectures by *adapting* the *behavior* of individual functions (see Req.3). This leads to different states of a function being associated with different situations and compositions of the CSG. Furthermore, the concept of *roles* allows restructuring of the CSG by means of the assignment of roles involved. Therefore, individual CESs conduct *role changes*, which influence the function and thereby again restructure the overall functional architecture.

4.6 Application of the Metamodel

In this section, we demonstrate the applicability of the metamodel using examples from the two use cases of the adaptable and flexible factory and autonomous transport robots.

4.6.1 Example from the Adaptable and Flexible Factory

For the adaptable and flexible factory, let us consider the scenario of order-driven production. In this scenario, there are several heterogeneous modules within a factory. These modules are equipped with different functions and can contribute to the production of products. Exemplary functions are drilling, milling, or even turning of materials. In addition, assembly operations can be used to assemble different workpieces or to execute optical quality checks. Depending on the product to be manufactured for a customer, different functions, and thus contributions from different modules are required. Those modules that can contribute to the production form a CSG in which they provide their functions to each other in order to achieve the overall goal of fulfilling the production order. At the end of the production, the modules leave the CSG again.

Centralized and decentralized coordination

Both centralized and decentralized coordination paradigms are conceivable for this CSG. In the case of centralized coordination, there is a single module that, as coordinator, decides after receiving a

production order which module is supposed to contribute to the production with which functions. In the case of decentralized coordination, the modules autonomously negotiate their possible contributions to the fulfillment of the order. Other mixed forms of centralized and decentralized coordination are also possible.

Regardless of the process of forming the CSG, the modules can be considered as CESs in accordance with the metamodel before the production starts and thus before the CSG is formed. By forming a CSG, these CESs become collaborating systems and, depending on their functional properties, assume one (or even several) roles in the CSG. Roles to be filled in the factory are, for example, material processing, assembly, transport, quality inspection and, in the case of centralized coordination, the coordinator. The required product can only be manufactured once all the roles required for an order have been assigned to the modules forming the CSG.

Modules assume roles

A module can only assume a role if it has the necessary functions. When all necessary roles have been assigned to modules, the CSG functions that are required to fulfill the CSG goals can be executed. An exemplary CSG function in the factory is the manufacturing of the product. The CSG function for manufacturing the product can only be executed, and thus the goal of fulfilling the customer order achieved, through the individual CES functions of the modules. Further exemplary CSG functions are the definition of the production sequence and the calculation of the production time. These CSG functions can also only be realized by aggregating the CES functions of the modules.

Separation between system function and collaboration function

The metamodel also shows a separation between system function and collaboration function. The system function represents the individual contribution of a CES to a CSG. A system function of a factory module can be drilling, milling, transport, or assembly, for example.

The collaboration functions enable the modules to communicate with each other, to exchange information about production orders, and to coordinate their contributions in the CSG. To coordinate the contribution of a module, the requirements of the products to be manufactured resulting from the orders must be compared with the available functions of the modules. In other words, a check is required to determine whether the functions of the modules are suitable to contribute to the production of the order. Such matching is also part of the collaboration functions.

www.dbooks.org

The coordinator matches requirements and functions

In a centrally organized factory, the task of matching the requirements of the order with available functions is the responsibility of the coordinator. In this scenario, each module that should be involved in possible production orders must inform the coordinator of its available system functions and provide appropriate descriptions of the scope of these functions.

Self-check contribution possibilities

In a decentralized factory, where the modules coordinate with each other without a central coordinator, each module must be able to check whether it can contribute to the production and must be able to communicate the result of this check to the other modules. The other modules must be able to understand this contribution and compare it with their contributions. This is the prerequisite for determining whether the contributions delivered in total (the resulting CSG functions) are sufficient to produce the product.

Receiving a new order within the factory can be seen as a trigger event from the context, which means that the modules have to adapt their behavior. The execution of this adaptation is enabled by the adaptation logic in the metamodel. A single module can adapt its behavior by using the adaptation logic. Such an adaptation can, for example, be that a module changes its current configuration and thus its executable functions. Depending on the specific module, this reconfiguration can be done automatically by the module itself or partially automated with the support of a worker in the factory.

4.6.2 Modeling of Goals for Transport Robots

Autonomous transport robots operate in a factory

Another example which helps to improve the understanding of the elements of the metamodel is based on the use case of transport robots. This example looks at several transport robots (i.e., CESs) within a factory, with each robot being responsible for transporting different materials (i.e., an overall CSG). In order to receive different materials as input for various products, the individual transport robots connect to modules and conveyor belts, which allows the transport robots to take part in multiple production processes at once. The main purpose (i.e., the system goals) of the transport robots consists of executing the production logistics and ensuring punctual delivery and pick-up of materials between production process sites, for which the transport robots provide several functions.

Negotiate orders and coordinate behavior

In order to optimize the transport of goods within the factory from a logistical point of view, individual transport robots must negotiate possible orders and distribute them jointly. To enable this negotiation and to coordinate further behavior, the transport robots must

collaborate. Therefore, several collaborative transport robots (CTR) form a collaborative transport robot fleet (CTRF). In the context of the metamodel, individual CTRs can be considered as CESs and the CTFR as the CSG.

Forming a collaborative transport robot fleet

By forming a CTRF, a CTR starts communicating to share information within the CTRF. This allows the CTRF to manage the operations of the CTRs. While a non-collaborative robot would typically optimize its own routes and transportations, the CTRF allows optimized utilization over all the CTRs. For more information on the close interaction between CTRs and the CTRF, refer to [Brings et al. 2019].

Goals of transport robots

In order to further illustrate the different goals of the CTRs and the CTRF, some of them are shown as an example in Figure 4-11. This figure models relationships between various goals and related tasks (i.e., specific functions to be implemented) and dependencies between the CTRs and a CTRF (i.e., the relationship between the functions of the CESs and the functions of the CSG). The modeling was performed using an extension of the goal-oriented requirement language (GRL) (cf. [Daun et al. 2019b], [Brings et al. 2020]). The goals that CTRs and CTRFs pursue are represented by curved boxes and they can be fulfilled by executing all connecting tasks, which are represented by hexagonal boxes.

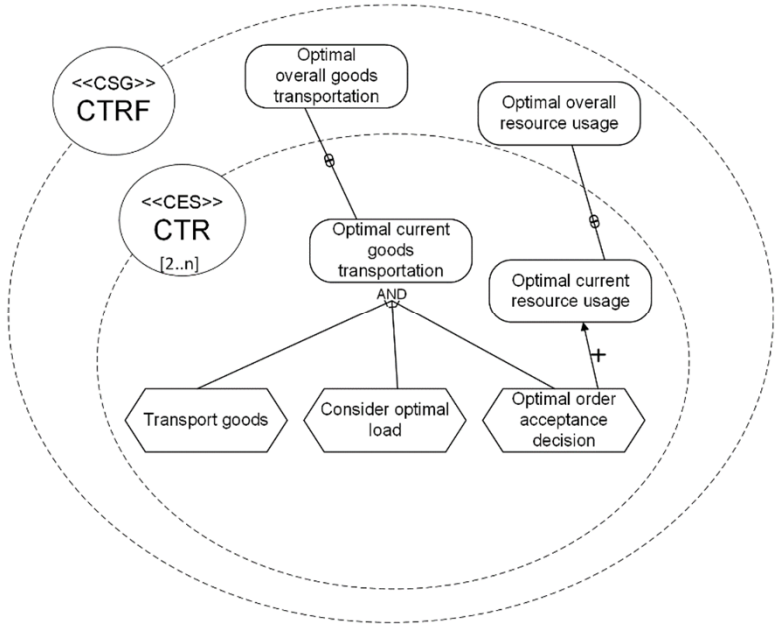


Fig. 4-11: *Goal model collaborative transport robot fleet*

www.dbooks.org

GRL goal modeling

Figure 4-11 shows an excerpt of the goal model for the CTRF. When applying GRL, these tasks can be divided into further tasks to allow a more detailed specification. In terms of the metamodel, these tasks can be considered as functions. The individual tasks of the CTR presented here correspond to system functions in the metamodel. The functions for communication between the CTRs within the CTRF that are not shown here correspond to the collaboration functions.

Differentiate between goals and tasks

The CTR pursues the goal to *optimize their current goods transportation*. The goal can be fulfilled when the CTR performs the different tasks shown. The CTRF pursues the goal to *optimize the goods transportation* of all participating CTRs. As these goals are interdependent, they are linked in the goal model by a bidirectional dependency (shown by the two *D*s on the connecting line). The task *optimal order acceptance decision* has some positive influence on the goal *optimal current resource usage* and is therefore displayed as a contribution arrow marked with the plus icon. While this refers mainly to the goal part of the metamodel, the relation to functions is made clear as the tasks define what functions need to be implemented to fulfill which goals.

4.7 Related Work

A comparative literature review was conducted by [Erden et al. 2008] to investigate different function modeling approaches and their similarities and differences. For example, functional model ontologies [Chandrasekaran and Josephson 2000], [Umeda et al. 1996], [Umeda et al. 1995], and [Yoshioka et al. 2004] aim at developing frameworks and languages for modeling the functionality of a system from the different viewpoints [Erden et al. 2008]. None of the proposed functional model ontologies consider the modeling of functions of complex CESs and CSGs, including the ramifications of the contexts in which these systems operate.

In [Chandrasekaran and Josephson 2000], the authors define two function viewpoints: “environment-centric viewpoint” and “device-centric viewpoint.” These viewpoints correspond to the collaboration and system functions proposed in our work respectively. In the first viewpoint, the function is related to the external effects that an object or a system has on its environment. In contrast, in the second viewpoint, functions are related to the internal features and parameters of the system. In our metamodel, to a certain extent we subsume both the viewpoints proposed in [Chandrasekaran and

Josephson 2000]: 1) considering systems' functions that have effects on their environments, specifically in our case the context as the relevant part of the environment; 2) as well as the other systems involved in a collaboration, including their internal system parameters, states, and behaviors. Furthermore, [Gero 1990] has developed a function–behavior–structure model. In his model, he considered a function as an intermediate step between the behavior of the system and the user's goal.

A few frameworks have been proposed in literature to define a well-formed functional behavior of the system systematically. FOCUS (cf. [Broy and Stølen 2012], and [Broy 2014]), previously explained in Section 4.3, is an instance of such a formal framework that provides models and formalisms for specifications and development of distributed interactive and dynamic systems. In our contribution, according to FOCUS, we define the behavior of a function as a stream of messages across its input and output channels, which through its interfaces, take up information, material, or energy, and transform it before outputting it.

To the best of our knowledge, there has been no previous work on modeling functions for CESs from multi-dimensional aspects as proposed in our metamodel, including the dynamicity of the contexts of the system, role and goal modeling, and complex properties of these systems such as collaboration and adaptivity. The domain-independent metamodel proposed in this paper closes this gap.

4.8 Conclusion

The new challenges in the model-based development of embedded systems arising from collaboration make it necessary to adapt and extend existing modeling languages. In this chapter, we showed the aspects to be considered in the modeling of functions for CESs and CSGs in a metamodel. We then evaluated this metamodel and illustrated it using two examples from the use cases of the adaptable and flexible factory and autonomous transport robots. Based on the metamodel, specific extensions of modeling languages can be executed. Depending on domain-specific requirements, methods for the application of these extended modeling languages can be developed. The use case examples presented in this chapter will be used as a basis for further research.

www.dbooks.org

4.9 Literature

[Brings et al. 2019] J. Brings, M. Daun, T. Bandyszak, V. Stricker, T. Weyer, E. Mirzaei, M. Neumann, J. S. Zernickel: Model-Based Documentation of Dynamicity Constraints for Collaborative Cyber-Physical System Architectures: Findings from an Industrial Case Study. In: Journal of Systems Architecture - Embedded Systems Design, vol. 97, 2019, pp. 153–167, DOI: 10.1016/j.sysarc.2019.02.012.

[Brings et al. 2020] J. Brings, M. Daun, T. Weyer, K. Pohl: Goal-Based Configuration Analysis for Networks of Collaborative Cyber-Physical Systems. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, Mar. 2020, pp. 1387–1396, DOI: 10.1145/3341105.3374011.

[Broy and Stølen 2012] M. Broy, K. Stølen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer Science & Business Media, 2012.

[Broy 2014] M. Broy: A Model of Dynamic Systems. In: From Programs to Systems. The Systems Perspective in Computing, pp. 39-53. Springer, Berlin, Heidelberg, 2014.

[Broy 2017] M. Broy: Formalizing Adaptivity, Dynamics, Context-Awareness, Autonomy, 2017, white paper.

[Chandrasekaran and Josephson 2000] B. Chandrasekaran, John R. Josephson: Function in Device Representation. In: Engineering with Computers 16.3-4: pp.162-177, 2000.

[Daun et al. 2019a] M. Daun, T. Weyer, K. Pohl: Improving Manual Reviews in Function-Centered Engineering of Embedded Systems Using a Dedicated Review Model. In: Software and Systems Modeling, vol. 18, no. 6, 2019, pp. 3421–3459, DOI: 10.1007/s10270-019-00723-2.

[Daun et al. 2019b] M. Daun, V. Stenkova, L. Krajinski, J. Brings, T. Bandyszak, T. Weyer: Goal Modeling for Collaborative Groups of Cyber-Physical Systems with GRL: Reflections on Applicability and Limitations Based on Two Studies Conducted in Industry. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019, 2019, pp. 1600–1609, DOI: 10.1145/3297280.3297436.

[Erden et al. 2008] M.S. Erden, H. Komoto, T. J. van Beek, V. D'Amelio, E. Echavarria, T. Tomiyama: A Review of Function Modeling: Approaches and Applications. Ai Edam 22, no. 2, 2008, pp. 147-169.

[Eisenbart et al. 2012] B. Eisenbart, L. Blessing, K. Gericke: Functional Modelling Perspectives Across Disciplines: A Literature Review. International Design Conference - Design 2012, Dubrovnik, Croatia, 2012.

[Gero 1990] J. S. Gero: Design Prototypes: A Knowledge Representation Schema for Design. AI magazine 11.4, 1990, pp. 26-26.

[Giese et al. 2013] H. Giese, H. A. Müller, M. Shaw, R. De Lemos: Software Engineering for Self-Adaptive Systems II. Springer, Berlin, 2013.

[Hayward et al. 2020] A. Hayward, M. Daun, W. Böhm, A. Petrovska, L. Krajinski, A. Fay: Modellierung von Funktionen in der modellbasierten Entwicklung von Systemverbünden kollaborierender cyber-physischer Systeme, Tagung: Entwurf komplexer Automatisierungssysteme (EKA), Magdeburg, 2020, (available in German only).

[Krupitzer et al. 2015] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, C. Becker: A Survey on Engineering Approaches for Self-Adaptive Systems. Pervasive and Mobile Computing 17, 2015, pp. 184-206.

[Lalanda et al. 2013] P. Lalanda, J. A. McCann, A. Diaconescu: Autonomic Computing: Principles, Design and Implementation. Springer Science & Business Media, 2013.

[Ludewig et al. 2018] A. Ludewig, M. Daun, A. Petrovska, W. Böhm, and A. Fay: Requirements for Modeling Dynamic Function Networks for Collaborative Embedded Systems. In: Joint Proceedings of the Workshops at Modellierung 2018 co-located with Modellierung 2018, Braunschweig, Germany, 2018, vol. 2060, pp. 79–89.

[Meissner et al. 2014] H. Meissner, M. Cadet, N. Stephan, C. Bohr: Model-Based Development Process of Cybertronic Products and Production Systems. In: Advanced Materials Research, Vol. 1018, 2014, pp. 539–546.

[Pahl et al. 2013] G. Pahl, W. Beitz, J. Feldhusen, K.H. Grote: Methoden und Anwendung erfolgreicher Produktentwicklung. 8. Aufl., Heidelberg: Springer. 2013 (available in German only).

[Petrovska and Pretschner 2019] A. Petrovska, A. Pretschner: Learning Approach for Smart Self-Adaptive Cyber-Physical Systems. *IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pp. 234-236. IEEE, 2019.

[Pohl 2010] K. Pohl: Requirements Engineering: Fundamentals, Principles, and Techniques. Springer Publishing Company, Incorporated, 2010.

[Pohl et al. 2012] K. Pohl, H. Hönninger, R. Achatz, M. Broy, Eds., Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology. Berlin Heidelberg: Springer-Verlag, 2012.

[Pohl et al. 2016] K. Pohl, M. Broy, H. Daembkes, H. Hönninger: Advanced Model-Based Engineering of Embedded Systems, Springer, Cham, 2016.

[Regnat et al. 2019] Regnat, N. et. al.: Seamless Model-Based Approach - MQ1.AP2.D4. Published by Subproject MQ1 in CrESt. Internal Project Deliverable of CrESt, 2019.

[Umeda et al. 1995] Y. Umeda, T. Tomiyama: FBS Modeling: Modeling Scheme of Function for Conceptual Design. Proc. of the 9th Int. Workshop on Qualitative Reasoning, 1995.

www.dbooks.org

[Umeda et al. 1996] Y. Umeda, I. Masaki, Y. Masaharu, Y. Shimomura, T. Tomiyama: Supporting Conceptual Design Based on the Function-Behavior-State Modeler. Ai Edam 10, no. 4: 275-288, 1996.

[VDI 2221] VDI Guideline 2221: Systematic Approach to the Development and Design of Technical Systems and Products, 1993.

[VDI 2222] VDI Guideline 2222 Blatt 1:1997-06: Methodic Development of Solution Principles, 1997.

[Vogelsang 2015] A. Vogelsang: Model-Based Requirements Engineering for Multifunctional Systems. Dissertation. Technische Universität München, Institut für Informatik, 2015.

[Weiß et al. 2019] S. Weiß et. al.: Modeling of Dynamics in the Open Context of Collaborative Embedded Systems – EC4.AP2.D3. Published by Subproject EC4 in CrESt. Internal Project Deliverable of CrESt, 2019.

[Yoshioka et al. 2004] M. Yoshioka, Y. Umeda, H. Takeda, Y. Shimomura, Y. Nomaguchi, T. Tomiyama: Physical Concept Ontology for the Knowledge Intensive Engineering Framework. Advanced engineering informatics, 18(2), 2004, pp.95-113.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Malin Gandor, OFFIS e.V.
Nicolas Jäckel, FEV Europe GmbH
Lorenz Käser, PikeTec GmbH
Alexander Schlie, TU Braunschweig
Ingo Stierand, OFFIS e.V.
Axel Terfloth, itemis AG
Steffen Toborg, PikeTec GmbH
Louis Wachtmeister, RWTH Aachen University
Anna Wißdorf, PikeTec GmbH

5

Architectures for Dynamically Coupled Systems

Dynamically coupled collaborative embedded systems operate in groups that form, change, and dissolve—often frequently—during their lifetime. Furthermore, the context in which collaborative systems operate is a dynamic one: systems in the context may appear, change their visible behavior, and disappear again. Ensuring safe operation of such collaborative systems is of key importance, while their dynamic nature poses challenges that do not occur in "classical" system design. This starts with the elicitation of the operational context against which the system will be designed—requiring capture of its dynamic nature—and affects all other design phases as well. Novel development methods are required, enabling engineers to deal with the challenges raised by dynamicity in a manageable way. This chapter presents methods that have been developed to support engineers in this task. The methods cover different viewpoints and abstraction levels of the development process, starting at the requirements viewpoint, and glance at the functional and technical design, as well as verification methods for the type of systems envisioned.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_5

www.dbooks.org

5.1 Introduction

Dynamically coupled collaborative embedded systems (CESs) have to function safely in collaborative system groups (CSGs) that form, change, and dissolve during the lifetime of the CESs. The members of a vehicle platoon, for example, typically change frequently. CESs and the corresponding CSGs must therefore be able to deal with internal dynamics as well as those of the operational context. Here, dynamics refers to a specific notion of the term that subsumes the following aspects:

Structure: the elements of the CES or CSG under consideration and their interaction and dependencies. For example, elements of the context can become part of the system group and emerge from it by leaving the group.

Function/behavior: the services offered by the CES or CSG, and the dependencies to the services in its context.

The above-mentioned aspects are indeed closely related. Systems form system groups in order to achieve overarching goals (as defined in Chapter 2). Vehicles, for example, may join a platoon in order to optimize space usage and traffic flow, which changes the internal system structure of the platoon. A car that drives in a platoon requires functions—such as certain coordination functions—that are different to those needed to drive independently. The functional aspect also concerns the visible behavior of the context, which may also dynamically change. CESs and CSGs must be able to change their behavior accordingly. In some application domains, such as in the traffic example, this aspect subsumes the perceived "intention" of other traffic participants.

Challenges addressed

This chapter focusses on three challenges that arise from dynamicity for the development of collaborative embedded systems. First, systems are typically designed against a context that impacts the definition of requirements, for example, the temperature range in which the system must be able to work. Defining such specifications becomes a complex task for dynamically coupled systems. The complexity results not only from the context dynamics, with changing context structures and behavior, but also from the system itself, which may dynamically become part of a larger system (group) and leave it again. At the end of this progression, we are faced with the problem of designing systems against open contexts that cannot be fully anticipated at design time.

Dynamicity also raises the challenge of managing design complexity. Starting with the functional design, how can we develop a functional architecture that reflects the dynamicity of the system context as well as the structure and behavior of potential CSGs in which the CES is intended to work? Dynamicity calls for novel architectural patterns, enabling engineers to deal with this kind of complexity. Finally, such architectures should also support validation and verification tasks — for example, by enabling compositional reasoning. As the class of systems considered is that of safety-critical systems, corresponding analysis methods that support engineers in assessing important safety properties should be applicable in a scalable way.

Chapter structure

This chapter presents methods that support engineers in designing dynamically coupled systems. The chapter is structured along the established design framework developed in the SPES projects [Pohl et al. 2012], [Pohl et al. 2016], as depicted in Figure 5-1. Section 5.2 introduces a contract-based modelling method for the specification of the behavior of collaborative system groups, covering collaboration and interface aspects of CSGs and their expected behavior. Section 5.3 elaborates on the functional design. The approach enables the modelling of refined function architectures with operation modes that reflect the dynamicity of context and system. Section 5.4 presents a novel approach for incrementally constructing system architectures that can function in dynamic contexts. Finally, Section 5.5 presents an analysis method for the safety aspect of collaborative systems at the logical design level. The analysis method allows assessment of the impact on safety of failures of the communication medium. The methods are exemplified in the context of the “Vehicle Platooning” and “Autonomous Transportation Robots” uses cases (cf. Chapter 1).

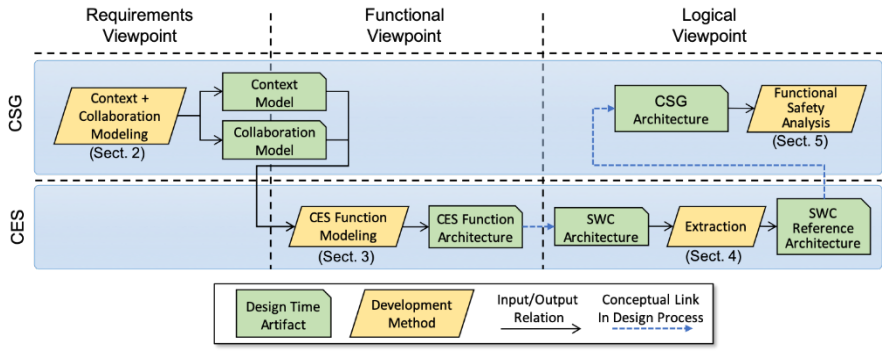


Fig. 5-1: *Method overview*

www.dbooks.org

5.2 Specification Modeling of the Behavior of Collaborative System Groups

This chapter introduces a modelling approach for a formal contract-based specification of collaborative open systems.

CSGs are formed by the CESs involved. While a CSG as a whole exposes behavior, follows its goals, and interacts with the environment, its behavior is actually implemented by the systems that make up the CSG. This implies that each system must be implemented correctly with respect to the required group behavior. To decide whether a CES fulfills its obligations in a collaborative system group, we choose the concept of contracts. Contracts, as presented in this approach, define the rights and obligations of the individual collaborative systems based on protocol state machines for peer-to-peer communication and formal scenario specifications of the group behavior. We want these contracts to be formal so that they can be used during CES operation but also already support automatic verification and simulation from a requirements perspective. This also implies that the modeling approach defines execution semantics so that specifications are executable.

Collaboration specification metamodel

The modelling approach covers different aspects that are relevant for specifying CSGs and the collaborative behavior of the CESs involved. The key concepts enable the scenario-based definition of collaboration structure and behavior. The metamodel in Figure 5-2 shows the main modelling concepts and their relationships, which are discussed in the following.

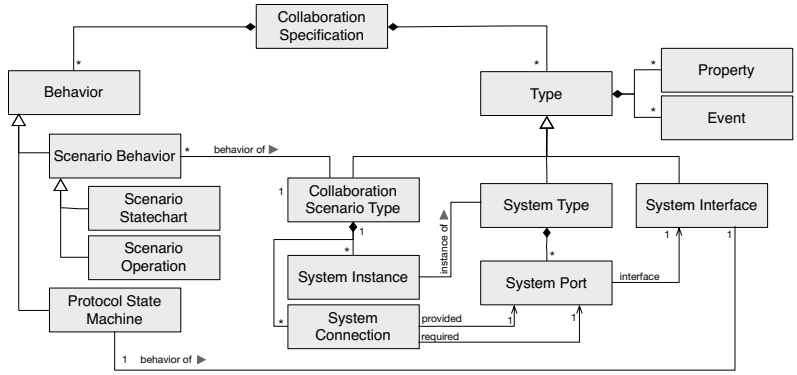


Fig. 5-2: *Collaboration metamodel*

These modeling concepts are implemented using a collection of integrated domain-specific modeling languages (DSLs). These consist of textual, grammar-based languages and the graphical notation of

statecharts. The concrete tools used are YAKINDU statecharts and slang (system language) [Yakindu 2019], together with Franca IDL [Franca 2019]. Independently of this concrete choice of modeling languages and tools, the underlying concepts can also be adapted to standard system modeling languages, such as SysML, or by proprietary modeling approaches. The concepts are exemplified by the "Collaborative Adaptive Cruise Control (CACC)" car platooning use case (see Chapter 1).

Collaboration scenario specification

The core approach for modeling collaboration within a CSG is based on formal specifications of scenarios. Scenarios constitute a natural way of specifying inter-object, or in our scope, inter-system behavior [Harel and Marelly 2003]. A CSG consists of a set of CESs and a set of relationships between these systems. This is specified by *collaboration scenario types*. The specification of such a type is illustrated by Figure 5-3. The example shows a platoon of three vehicles that form a CSG. Each CES involved is represented by a *system instance* of *system type PlatoonMember*. The direct communication relationships between the CESs are specified as *system connections*.

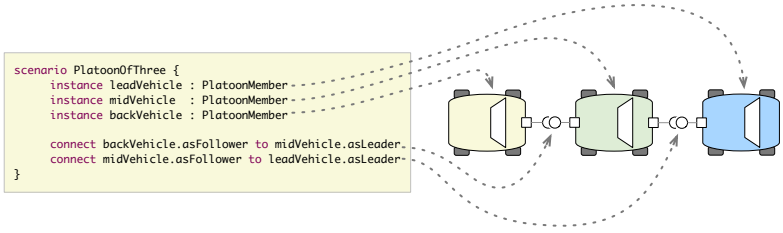


Fig. 5-3: *Example CSG structure*

For this type of collaboration scenario structure, a specification defines a set of behaviors. In contrast to other scenario specifications, such as use case descriptions or standard sequence charts, CSG specifications following this approach must be executable and thus require a high degree of formalism. To support this, two types of behavior models are used: *scenario operations* and *scenario statecharts*. A *scenario operation* is a simple procedural model for specifying dynamic processes within a CSG. Figure 5-4 gives an example of a CSG reconfiguration within the vehicle platoon that adds and integrates vehicles.

www.dbooks.org

```
@scenario op joinToSingleLead() {

  // first place a car into the scenario
  midVehicle.location = Coordinate.new(0, 0)
  midVehicle.velocity = 50
  assert notConnected( midVehicle.asFollower )
  // let the time proceed without creating a platoon
  time.proceed( minutes(5) )
  assert notConnected( midVehicle.asFollower )
  assert ( midVehicle.location.X == 50*60*5 )

  // place second car 200 meters in front of first car
  leadVehicle = PlatoonMember.new
  leadVehicle.location = Coordinate.new(midVehicle.location.X + 200, 0)

  leadVehicle.velocity = 40

  // as soon as the first car comes close to the second car
  // the platoon will be established
  time.proceedUntil( leadVehicle.location.X - midVehicle.location.X < 100 )
  assert ( midVehicle.asFollower == leadVehicle.asLeader )

  // after some time the platoon is cruising with the second car
  // velocity and constant distance.
  time.proceed( seconds(20) )
  assert (midVehicle.velocity == leadVehicle.velocity )
  assert (leadVehicle.location.X - midVehicle.location.X == 55 )
}
```

Fig. 5-4: *Reconfiguration example for platoon creation*

CSG reconfigurations apply changes to the coupling of CESs and are thus a way to capture the dynamics. Basically, all modifications such as adding, removing, connecting, and configuring CES instances can be described. Moreover, time is an explicit concept that can be used to control temporal aspects of the scenario. Finally, assertions check the proper execution of a scenario.

Scenario statecharts (introduced in [Marron et al. 2018]) adapt the concepts of scenario-based modeling (SBM). SBM is an approach that was first presented in the form of the graphical formalism of *life sequence charts (LSC)* [Damm and Harel 2001], [Harel and Marelly 2003]. Scenario-based statecharts extend the formalism of statecharts [Harel 1987] with SBM concepts. A *scenario statechart (SSC)* (see Figure 5-2) describes a scenario that covers a single behavioral aspect of the system group. Different scenario statecharts can be combined to obtain a behavioral description of the system group. The synchronization between these scenarios is based on events. In each state, an SSC can *request* or *block* events. All events that are *requested* by at least one scenario and are not *blocked* by at least one other scenario are called *enabled*. One or more enabled events can be selected and activated by a central event selection mechanism. All

scenarios that requested or waited for such an event will be notified and can proceed to the next scenario state.

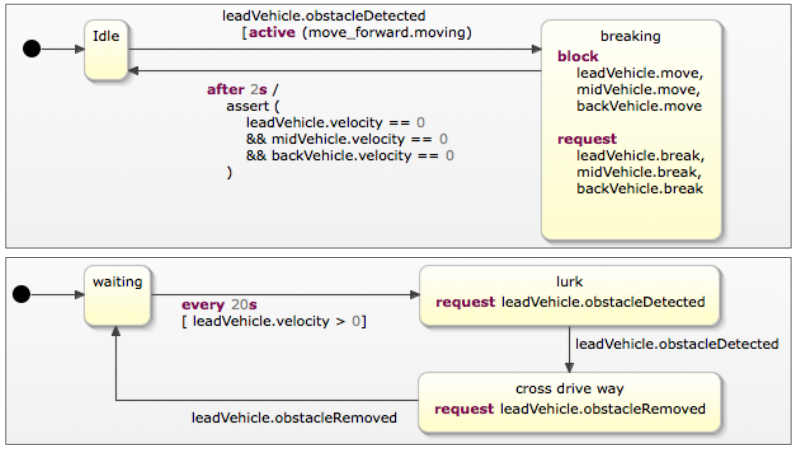


Fig. 5-5: *Emergency stop and obstacle detection scenarios*

Figure 5-5 illustrates two example scenarios, each defined using a simple *scenario state chart*. The first specifies an emergency stop based on an obstacle. The second specifies the obstacle detection. Both refer to the platooning CSG but are not directly dependent on each other.

System type specification

All properties of a CES that are relevant for the CSG specification are specified by a *system type*. As an example, *PlatoonMember* (Figure 5-6) reacts to incoming events and defines a set of system properties such as *velocity* and *frontDistance*. It also defines direct collaboration relationships to other vehicles using *system ports*. The system port *asLeader* provides a *CACControl* interface and *asFollower* requires it.

System interface specification

CACControl is a *system interface* that defines the elements that can be used in the interaction (or communication) between two systems. This concept adapts the well-known concepts of interface and protocol specifications, as the modeling approach assumes that communication protocols will form the basis for inter-CES communication.

www.dbooks.org

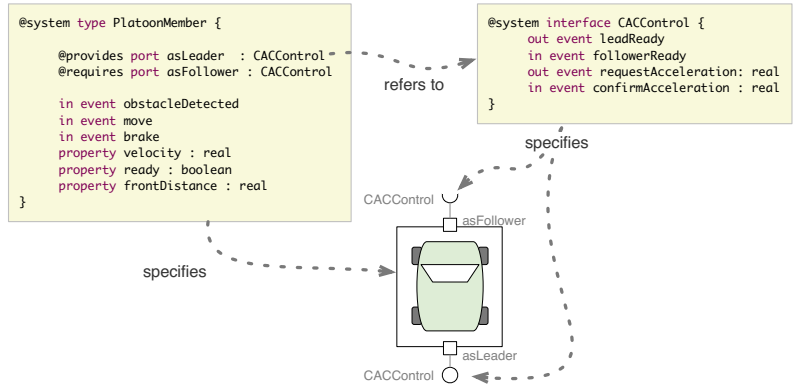


Fig. 5-6: *System type and system interface example*

System interface contracts

System interfaces define the elements that exist in the interface and are used by the interaction of CESs. The proven concept of protocol state machines (PSMs) [Franca 2019] allows specification of the dynamic behavior of *system interfaces* and can be used to ensure that the communication peers involved interact in the proper order.

CSG contracts validation

The behavioral part of a CSG *collaboration specification* is made up of all *scenario operations*, *scenario statecharts, and PSMs*. The scenario-based modeling approach is inherently incremental, which involves incremental specification, development, and integration of dynamically coupled CSGs and CESs. Additionally, all behavior models are inherently executable. All models described can be jointly executed within a simulation without any further behavioral model. This already serves as a basis for analysis methods that check the properties and consistency of the specification itself. Moreover, if the specification models of the CSG are executed together with the behavioral models of the CESs (e.g., using co-simulation), then conformity and consistency of the CESs with the CSG specification can be checked automatically. This allows a complete specification of the collaborative behavior of a CSG for all known aspects in a dynamic context, which is a precondition for the verification of the CES behavior within a CSG. Comparable to PSMs that define an interaction contract for single interaction relations, the collaboration scenarios defined by a CSG specification form the *collaboration contract* for all CESs involved.

5.3 Modeling CES Functional Architectures

The functional architecture of a CES establishes the link between the requirements viewpoint and the system design (cf. Figure 5-1). A functional architecture "integrates the system requirements in a structured, implementation independent system specification" [Pohl et al, 2012]. It should therefore reflect all aspects discussed in Section 5.2, including dynamicity. The basic idea of the modelling approach presented in this section is to explicate relevant system states in the functional architecture model in order to enable consistency to be established between the functional model and the dynamic aspects of the CSG specification — that is, the functional design of the individual CESs realizes the dynamic aspects specified in the requirements viewpoint.

Functional architecture metamodel

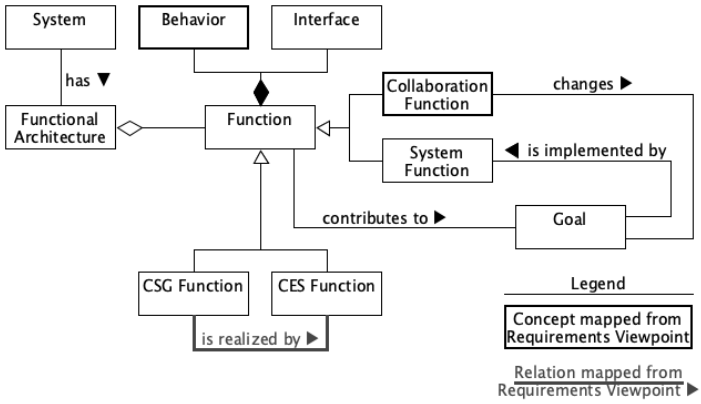


Fig. 5-7: *CrESt functional architecture metamodel — excerpt*

The approach conforms to the metamodel defined with the SPES modelling framework [Pohl et al. 2012], which has been extended in CrESt in order to reflect the need to design (dynamic) collaborative systems as well (cf. Chapter 4). An excerpt of this metamodel is depicted in Figure 5-7. It reveals the relationship between the concepts discussed in Section 5.2 (bold boxes) and the functional elements. The specified system behavior, for example, will be allocated to the *behavior* of a function. The collaborative behavior specification (cf. Figure 5-5) is allocated to *collaboration functions* of the CESs, while the collaboration structure and their relationships determine the way in which *CSG functions* are *realized* by *CES functions*. The figure also shows the relationship of the *functional architecture* to the *goals* a CES or CSG is aiming for.

www.dbooks.org

Functional architectures for dynamic systems

Modelling functional architectures of dynamic systems requires paying particular attention to the relationships between system functions and goals. As introduced in the Chapter 2 collaboration functions determine the goals a CES (or CSG) is following at a particular point in time. The goals in turn are *implemented* by the system functions of the individual CESs. Dynamic changes in the CES (and CSG) are reflected by changes in the *collaboration functions*, and in turn in the *system functions*. The dynamic interplay between goals and functions requires changes to happen in an orchestrated way. The individual system functions must switch their internal behavior consistently in order to be able to contribute to the changing goals. The proposed modelling approach allows the specification of such functional dynamics in terms of state diagrams, where engineers can explicate the dynamicity of functional behavior and the interaction between functions to coordinate changes. The approach then enables analysis of whether dynamic changes actually happen in a consistent way with respect to the scenarios specified.

5.3.1 Scenario

System scenario — example

The approach is exemplified by the "Autonomous Transport Robots" use case (cf. Chapter 1). Figure 5-8 shows a simple scenario with a single production machine and two transport robots, which represent the CSG being designed. Each robot is a CES in this CSG. The goal of the CSG is to transport goods between machines as well as storage locations, following some optimization objectives (cf. Chapter 9). Transport requests from the machines are distributed among the individual robots.

The scenario specification in Figure 5-8 is similar to the one introduced in Section 5.2 but applied to a different use case. The scenario consists of a simple sequence of snapshots that represent a particular state of the system and its context. Both robots initially do not perform transport tasks. This is indicated by a *wait* state assigned to the robots. In the second step of the scenario, the machine issues a *request* for a transport *task*, such as the delivery of a required resource, or the pickup of goods produced by the machine. This state is depicted on the right-hand side of the figure. The third step in the scenario specification would be that one of the robots (here *robot2*) takes over responsibility for the task.

This type of scenario typically also consists of a specification of the interaction between the individual objects, such as sequence charts defining messages that are communicated, causing a scenario to transition from one snapshot to another. In our scenario, this is exemplified by single events. In Figure 5-8, the events are written in boldface. For example, the scenario transitions from the first to the second snapshot as a result of the occurrence of a *newTask* event.

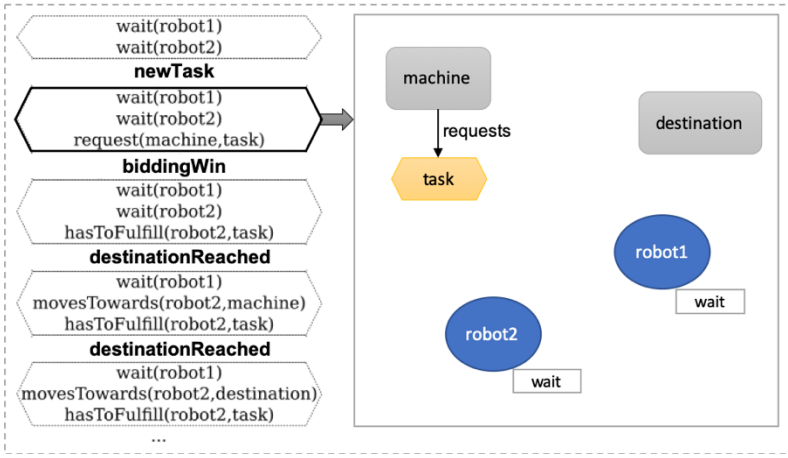


Fig. 5-8: *Autonomous transport robots use case — example scenario*

The scenario actually exhibits the dynamic nature in the context of the CSG "Transport Robots." Although a transport task is not a physical entity, it corresponds to a transported product as a physical object that appears in the context of the transport robots. Products and other relevant dynamic aspects, such as temporary roadblocks and the addition of robots to the fleet, have been omitted to keep the discussion simple.

5.3.2 Modelling

The modelling approach as depicted in Figure 5-9 is consistent with [Vogelsang 2015] and employs the concepts for a structured mapping of the collaboration specification to the functional architecture. The top part shows the functional architecture of a transport robot, consisting of the functions *Planning & Control*, *Bidding,* and *Dynamic Control*. The key element of the mapping is shown in the angled boxes. They represent the system states and object relations derived from the specification, which are relevant for the individual functions. The *Bidding* function realizes the collaboration among all robots by

www.dbooks.org

negotiating which robot takes over a transport task, and therefore decides about the *hasToFulfill* relationship of a task. The *Dynamic Control* function is responsible for navigating the robot safely through the factory, and thus realizes states such as *wait* and *movesTowards*.

The bottom part of Figure 5-9 shows the realization of the functions in the logical architecture. It has been modelled in terms of a SysML Internal Block Diagram, which has been chosen as the implementation language. The *Planning & Control* function maintains the "global" state of the transport robot. Figure 5-9 also shows how the interactions between the individual functions are realized, modelled by events that are transmitted between the interfaces along the connections. For example, an incoming *newTask* event to the

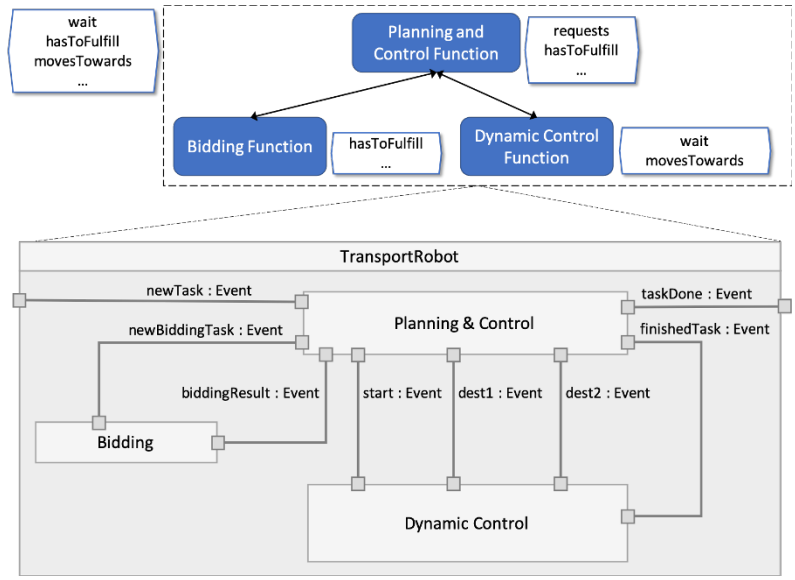


Fig. 5-9: *Robot top-level functional architecture (top), and its realization in the logical viewpoint*

Planning & Control function causes a request to the *Bidding* function, which will eventually come back with the result of the respective bidding process. This in turn causes the *Planning & Control* function to request state changes of the *Dynamic Control* function in order to perform the required operations.

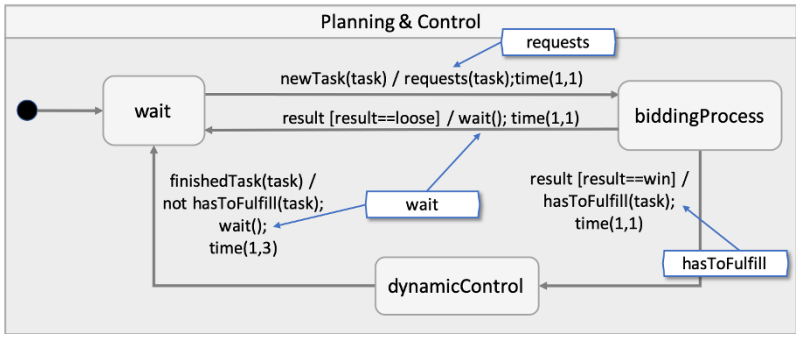


Fig. 5-10: *Planning & control — state-machine diagram*

Typically, a large number of scenarios are specified for reasonably complex systems and contexts. Moreover, the individual functions will be further decomposed along the modelling process. Supporting engineers in ensuring that the architecture designed adheres to the requirements specified in the scenarios is of crucial importance in order to avoid design errors. Figure 5-10 shows how this can be achieved with the proposed modelling approach by explicating internal state changes of the individual functions in terms of SysML state machine diagrams. While state machines are defined mainly to model behavior, they also provide a natural way to specify dynamicity in functional architectures and the interaction between functions in order to coordinate state changes throughout the CES architecture.

However, relating individual states with the system states specified in the scenarios, would require a great deal of effort and becomes highly complex— for example, if only combinations of states of different functions match particular scenario states. A more convenient and suitable way is to identify interaction points, or more precisely, transitions, in the state machines, with corresponding state changes in the scenarios. This is shown in Figure 5-10. The angled boxes denote the events (and in turn state transitions) that are associated with establishing object relationships in the scenario specification.

As SysML state machines provide a large number of features, a small subset of them have been selected and some design rules have been defined to make the approach effectively applicable. More details about this can be found in [CrESt 2019].

www.dbooks.org

5.3.3 Analysis

Consistency analysis

The section concludes with a brief overview of an automated analysis that can be applied in order to check the consistency of the functional architecture modelled with a scenario specification. To this end, both the scenario specification and the functional architecture, including the state machine diagrams, are automatically translated into a target automaton model (in our case RTana$_2$ [Stierand et al. 2016]). The translation has to identify state changes by events as explained above. In the current implementation, this is achieved by name matching. The analysis is basically a refinement check that fails if the architecture model cannot "follow" the scenario specification, that is, where either expected events do not occur (e.g., a *hasToFulfill* event of one robot), or events occur unexpectedly (*hasToFulfill* events from multiple robots). Note that consistency analysis has been developed in the context of all SPES projects, such as with the AutoFOCUS tool [Pohl et al. 2012, Section 5.5]. We now apply this important analysis step to dynamic systems.

5.4 Extraction of Dynamic Architectures

Reference architectures can be used to define common structures in software product lines for CES engineering. Therefore, they determine the static and dynamic compositions of the underlying software architecture. Reference architectures can either be defined from scratch or extracted from a set of system architectures for specific contexts expected for the CSG. Extraction enables identification of existing features through successively establishing a reference architecture by analyzing system architectures. The extraction process captures the commonalities and variations of the architectures analyzed. For that reason, the reference architecture forms the basis for the development of further products and can be successively extended by the extraction process.

The methods we present for extracting reference architectures from a set of architecture models is semi-automated. Logical system architectures for a static context are developed upfront and extrinsic matches (common parts in each architecture) with the current reference architecture are identified automatically in a second step. All components of static system architectures that do not match extrinsically in the reference architecture are automatically assigned to the reference architecture. To minimize the number of false assignments, this assignment is then reviewed by a domain engineer

manually. The remaining extrinsic matches are further analyzed to identify differences. For this purpose, fully automated variant and similarity analyses are performed during the extraction process.

We begin this section by introducing general principles of software product line engineering and continue with an explanation of how new domain artifacts can be derived from the bases of multiple application artifacts. As these techniques rely strongly on the establishment of reference architectures, this section concludes by introducing the Family Mining [Wille et al. 2014] approach, which provides mechanisms for establishing reference architectures based on a set of architectures that already exist.

5.4.1 Methods

To extract dynamic system architectures from existing system architectures, this section is structured as follows. First, we introduce reference architectures, which describe the common structures of product lines. Second, we use the concept of software product lines, for which we present a product-driven approach. Finally, we discuss the extraction with the Family Mining approach in the context of employed methods, that is, the Static Connectivity Matrix Analysis (SCMA) [Schlie et al. 2018] and the Reverse Signal Propagation Analysis (RSPA) [Schlie et al. 2017], which are both explained in detail below. Clone-and-own [Riva and Rosso 2003] is a straightforward reuse strategy that describes the copying and subsequent modification of an existing system to create a new system variant.

With regard to software architectures, this straightforward reuse strategy leads to a vast quantity of redundant and similar artifacts. Moreover, a later transition towards structured reuse, such as with software product lines, inevitably requires the comparison of all existing variants prior to the actual migration. The development of dynamic open systems from scratch adds a new level of complexity to the system as it involves designing new functionality at the same time. Thus, a step-by-step development based on a specific context of the CSG by reusing a common reference architecture is promising. In this process, the common parts of the system are reused in the reference architecture of the system, while new parts represent the dynamic part of the system.

Static Connectivity Matrix Analysis (SCMA)

SCMA [Schlie et al. 2018], [Schlie et al. 2019] is a procedure that enables the evaluation of multiple MATLAB/Simulink model variants simultaneously. The transformation of models into a matrix form reduces the complexity of the models and allows large-scale systems

www.dbooks.org

to be compared with each other in their entirety. Moreover, SCMA identifies all similar structures between the system portfolio under comparison, even with model parts being completely relocated during clone-and-own.

Reverse Signal Propagation Analysis (RSPA)

During development, model-based systems are subject to frequent modifications. Manual identification of all modifications performed is typically not feasible, especially for large-scale systems. However, precise identification and subsequent validation of the modifications is essential for the overall evolution. RSPA is a procedure that identifies and clusters variations within evolving MATLAB/Simulink models.

With each cluster representing a clearly delimitable—i.e., separate—variation point between models, model engineers can not only specifically focus on single variations, but by using their domain knowledge, they can relate and verify them.

Family Mining

One of the main challenges in the development of dynamic architectures is capturing changes in the system's context and subsequently adapting the system to adjust to these changes. Thus, the resulting architecture must allow a dynamic reconfiguration in response to a changing context of the CSG. To this end, dynamic software components of the architecture may only be relevant for a set of contexts, and therefore, multiple alternative implementations of a component may exist.

Software product line engineering (SPLE)

Software product line engineering (SPLE) deals with similar challenges. In SPLE, software components or software modules are flexibly configured to different application scenarios. Different binding times, that is, the times of selecting and deriving the concrete software variant of these modules are possible — for example, configuration time, compilation time, initialization time, or runtime. Dynamic open system architectures can be seen as software with a binding time at runtime. Consequently, development mechanisms of SPLE can be applied to the development of flexible system architectures.

5.4.2 Software Product Line Engineering

A software product line (SPL) enables software developers to tailor their software products to individual customer needs [Clements et al. 2001], [Apel et al. 2013]. To this end, an SPL captures the commonalities and variabilities of a given set of software systems and derives concrete software products by means of a variant deviation mechanism. This mechanism takes a collection of desired software

functionalities, called a *configuration*, as an input and automatically derives a software variant from the SPL.

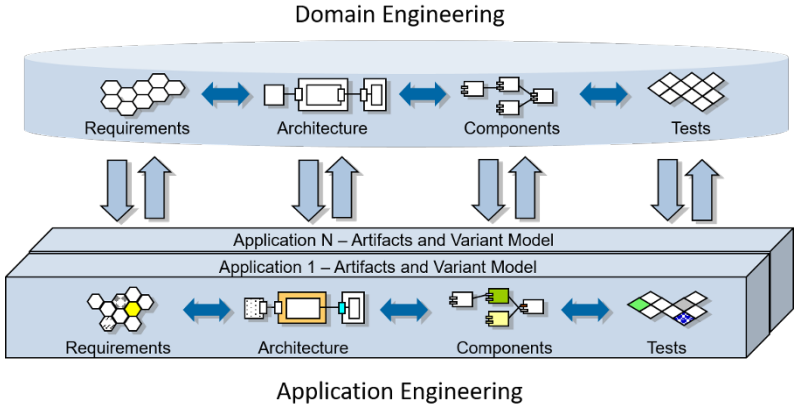


Fig. 5-11: *Reactive product line engineering (based on [Pohl et al. 2005])*

As for reference architectures, there are extractive methods for SPLE as well as proactive approaches that aim to establish an SPL from scratch. Reactive SPLE [Apel et al. 2013] aims to combine the strengths of both approaches. The aim of this process is to handle the fact that products might be added to the SPL in later phases of the product life cycle, or that specific software variants are altered after their derivation, which often occurs in practical applications. To achieve this aim, the reactive SPLE as displayed in Figure 5-11 starts with an initial SPL, which consists only of a basic set of products that is created from scratch, and later uses extractive mechanisms to evolve the SPL and incorporate changes to the requirements and product variants — that is, that existing products may be altered, or new products may be included [Apel et al. 2013].

5.4.3 Product-Driven Software Product Line Engineering

Product-driven software product line engineering is a form of reactive SPLE that focuses on the step-by-step establishment and development of a software platform based on established artifacts considering new requirements arising from application engineering. Using an extractive approach, new domain artifacts can be derived from the basis of multiple application artifacts. The process for developing a new software component variant using the product-driven SPLE

www.dbooks.org

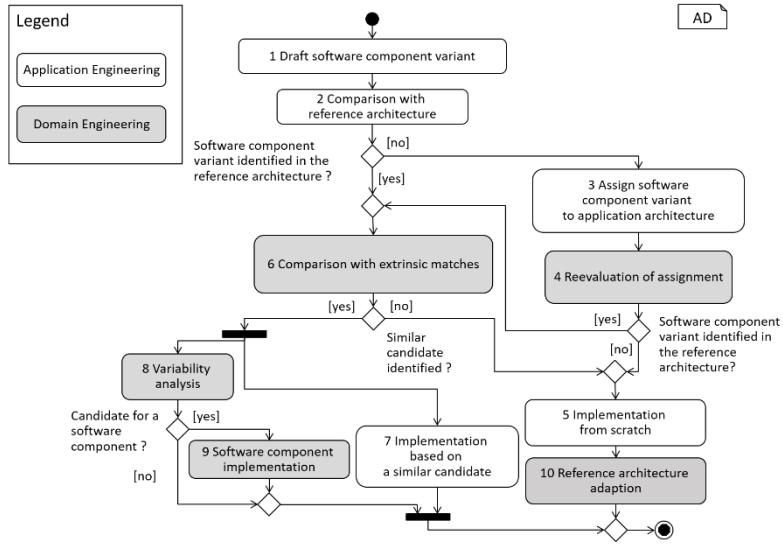


Fig. 5-12: *Product-driven software product line engineering*

approach is illustrated in Figure 5-12 as an activity diagram (AD) and consists of the following steps:

1. The “Draft software component variant” activity provides a name and a short functional description.
2. The check whether the functionality fulfilled by the software component variant is also fulfilled by a software component of the reference architecture is done in the “Comparison with reference architecture” activity. If this is the case, the names of both components should be identical.
3. If the software component variant identified in the reference architecture can be assigned to the application architecture, the activity “Assign software component variant to application architecture” will do this.
4. If the current software component has no counterpart in the reference architecture, the “Reevaluation of assignment” activity requires that the domain engineers recheck the assignment again
5. If no software component variant is identified in the reference architecture, no synergies can be provided by the current software platform, and thus an implementation from scratch is necessary in the “Implementation from scratch” activity.
6. The activity “Comparison with extrinsic matches” analyzes the similarity of the components based on structural and

semantic aspects of the extrinsic matches to identify commonalities and differences between the software components.

7. A similar implementation based on this candidate is possible and performed if a similar candidate exists. This is done in the activity "Implementation based on similar candidate."
8. Commonalities and differences can be analyzed in detail in the "Variability analysis" activity to identify possible variation points and variants, if similar available software component variants can be identified.
9. Based on the variability analysis, the "Software component implementation" activity includes the creation of a new software component such that its configuration matches its extrinsic matches.
10. The activity "Reference architecture adaption" includes the adaption of the reference architecture to incorporate a new component.

5.4.4 Family Mining — A Method for Extracting Reference Architectures from Model Variants

Coarse-grained analysis

To extract variability relations between existing block-based model variants, such as MATLAB/Simulink models or SysML statecharts (cf. [Alalfi et al. 2014], [Font et al. 2015], [Martínez et al. 2014], [Nejati et al. 2007], [Rubin and Chechik 2012], [Rubin and Chechik 2013a], [Rubin and Chechik 2013b], [Ryssel et al. 2010], [Ryssel et al. 2012]), the Family Mining approach was developed [Wille et al. 2014]. The approach provides a generic algorithm that is not only applicable to different block-based modelling languages, but also enables customization by providing user-adjustable metrics [Wille et al. 2016], [Wille et al. 2018].

To present the workflow of this Family Mining approach, Figure 5-13(a) depicts the steps required to compare input systems, locate the

www.dbooks.org

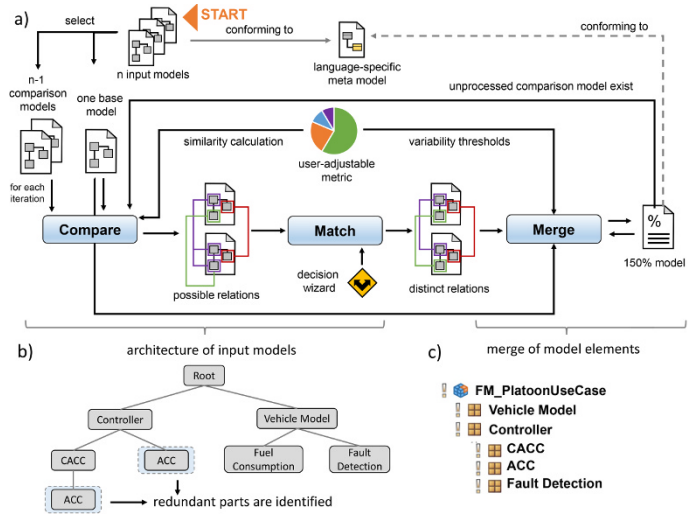


Fig. 5-13: *Workflow of the custom-tailored Family Mining approach for identifying variability relationships between block-based model variants*

most similar elements to match them with one another, and to derive a 150% model starting from a set of imported input models.

Fine-grained analysis

Moreover, Figure 5-13(b) illustrates how the approach can be used to capture the systems' underlying architecture by assessing all input models (i.e., the entire portfolio) at once [Schlie et al. 2018]. Hence, the structural components (here MATLAB/Simulink *subsystems*) of the input systems, along with their hierarchical relationships, are assessed and related to derive the overall architecture of the input portfolio and to simultaneously capture redundant model parts (cf. *ACC* in Figure 5-13(b)). Subsequently, the workflow shown in Figure 5-13(a) can be applied in a fine-grained fashion to only those components warranting such analysis, for instance to locate variation points at a fine level of detail [Schlie et al. 2017] and to derive a final 150% model [Schlie et al. 2019]. Such a 150% model (cf. Figure 5-13(c) for an excerpt) contains all possible model elements with annotations to indicate where variants' respective elements originated from and variation points between them. To extract such variability information and represent it in a centralized form, meaning the 150% model, the workflow evaluates block-based model variants in three sequentially processed phases (cf. compare, match, and merge in Figure 5-13(a)).

Compare

In the first phase, called the compare phase, the identification of model relationships is the primary goal of interest. For this purpose, the imported model instances are compared with each other. The workflow allows for variants to be compared at different levels of

Similarity

granularity and using different techniques. First, systems can be compared iteratively, selecting a base model (e.g., the smallest model) and processing the remaining n-1 models iteratively, each further model variant then serving as a comparison model for the current comparison phase. In this phase, the structure of the block-based input models with their nodes (e.g., functional blocks for MATLAB/Simulink systems) and their directed edges (e.g., signals used to relay data between nodes) is exploited. To compare the nodes of the input model, the proposed workflow starts with the start nodes of the models (e.g., input blocks that introduce data) and traverses nodes following the direction of data flow and, at all times, compares nodes based on the user-adjustable similarity metric. This metric calculates a similarity value in the interval [0..1], with 1.0 indicating 100% similarity. This similarity value is stored in a comparison element, along with the elements being compared and their possible relationship within analyzed models under comparison. Next, the traversal algorithm follows the outgoing edges of the node and compares them until no further compared nodes can be found. Another technique offered by the workflow, SCMA [Schlie et al. 2018], abstracts from the models' inherent graph structure and describes the models in a matrix form, representing only salient system information, as described below. With models being structured in a hierarchical fashion, with each hierarchical element denoted as a subsystem in MATLAB/Simulink, each subsystem is transformed into matrix form separately. As a result, the overall complexity of such model-based systems is reduced drastically, allowing for the comparison of multiple systems at once, rather than in an incremental fashion. This allows system parts that warrant a fine-grained analysis to be identified. Hence, such fine-grained analysis can be employed only when warranted, omitting unnecessary comparisons.

A more fine-grained comparison procedure, RSPA [Schlie et al. 2017], compares block-based systems by assessing changes between individual signals that always connect two blocks and grouping affected blocks into delimitable variation points. In contrast to SCMA, RSPA compares exactly two models, and can therefore be integrated within the iterative comparison of an entire system portfolio. Like SCMA, RSPA identifies areas within models where variations exist, allowing for a precise targeting of such parts in the context of the overall workflow.

Match

In the second phase of the workflow, the matching phase, the elements that are the most similar are matched with one another and are assigned with their specific relationship (i.e., their variability),

www.dbooks.org

based on their similarity value. Multiple possible matching partners may exist for a distinct element (e.g., a block from one model being compared with multiple blocks from a different model). Such ambiguities are identified and resolved during matching. Here, the matching algorithm analyzes the comparison elements from the "compare" phase and checks whether other comparison elements that comprise one of the contained model elements exist. In this case, the matching element with the highest similarity value is chosen. If both compared elements have the same similarity value, these comparison elements are sorted to the end of the list and the algorithm tries to solve the conflict by matching other comparison elements first. If the conflict remains, a *decision wizard* is called to identify the desired match by executing additional user-specified logic or by requesting direct feedback from the user.

In contrast, SCMA explicitly utilizes comparison results from multiple input models to determine similarities across system boundaries and across respective locations therein. Relating similar comparison elements from multiple models to one another, while exploiting the hierarchy of compared elements, allows information about the model portfolio being analyzed to be retrieved. Moreover, redundant or highly similar functionality, which may reside at different locations within systems, can be identified. Such redundancies can then be processed separately prior to the final phase, the transformation of compared artifacts and their relationships within a centralized form.

Merge

In the third and final phase, called the "merge" phase, the merge algorithm creates a 150% model to store the variability relationships identified. To this end, the algorithm extends a copy of the base model by merging all matched components into this model. Based on the similarity values from the "compare" phase, the algorithm determines the explicit variability by categorizing the elements into mandatory parts (i.e., common parts of all models), optional parts (i.e., common parts of some models), and alternative parts (i.e., mutually exclusive parts between models). During this process, all model elements that were not previously part of the base model are copied to the 150% model.

This 150% model generated enables domain experts to analyze the variability identified in detail. Moreover, it may serve as a basis for the comparison of the next remaining comparison model. The proposed algorithm thus iteratively compares and merges all input models into a single 150% model that stores the variability information for the model family analyzed.

5.4.5 Summary

In summary, SPLE enables software engineers to capture commonalities and variabilities of a given set of software systems and to derive concrete software products by means of a variant derivation mechanism during CES engineering. To combine the strengths of creating SPLEs from scratch with the advantages of extractive SPLE, the reactive product-driven SPLE approach describes a step-by-step establishment and development of a software platform based on established artifacts. The Family Mining approach starts with input models, which are first subject to a coarse-grained analysis, denoted SCMA. In the SCMA, similar parts that warrant further analysis are identified, while identical (meaning redundant) parts within models are eliminated. By omitting unnecessary comparisons, the Family Mining approach then directs subsequent analysis procedures to those similar parts. Specifically, we employ a fine-grained comparison metric to capture the variability of individual model elements at fine-grain level (e.g., varying labels or different internal properties). Comparison results of the fine-grained analysis are combined with information from the coarse-grained analysis to derive one holistic 150% model.

5.5 Functional Safety Analysis (Online)

A common way to ensure the correct functional behavior of an existing system is systematic testing against requirements. This testing usually occurs with a model or setup of the system that is already running instead of an architectural model. Therefore, we call this testing online analysis with regard to functional safety. If the system under test (SUT) is a CSG, there are further safety-relevant requirements regarding the collaboration. These cannot be properly tested with just a single CES as the SUT.

As described in Section 5.2, the entire idea of collaboration between different CESs is highly dependent on communication. If the communication is faulty, no collaboration is possible. A single CES should still be able to react when faced with faulty communication. Therefore, the recognition of faulty communication is an important situation that must be tested.

For this purpose, we have developed a method to inject communication errors into a CSG as the SUT. This allows faulty communication to be simulated deterministically to test and verify various kinds of error-detection mechanisms.

www.dbooks.org

For evaluation purposes, we implemented this method with AUTOSAR components as an example. The result is a prototypical test environment that connects multiple AUTOSAR components. This environment enables us to intercept the communication between components and manipulate the data exchanged.

5.5.1 Functional Testing

Software development for embedded systems typically starts with the specification of the desired behavior. Such specifications often contain expectations of output signals considering certain input signals. For example, "If the distance to the car in front falls below 100 m then the brake must be applied" could be a basic specification of an emergency brake system. The scenario statecharts introduced in Section 5.2 can also serve as a specification of the behavior. The software is implemented based on such specifications.

To test an implementation, the software must be stimulated with input signals and the output signals must be recorded. The device that stimulates the inputs and records the outputs is called the test driver. The tracking and evaluation of those signals against functional requirements is called functional testing. A schematic representation of this basic procedure for software testing can be seen in Figure 5-14.

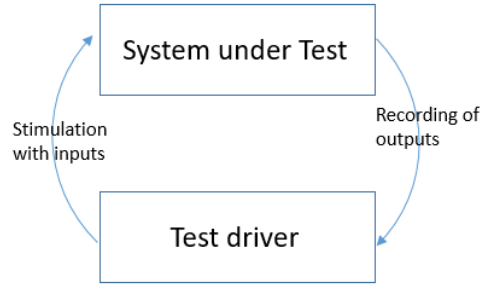


Fig. 5-14: *Software simulation — schematic representation*

Test solutions connect a test driver with the CES or CSG to be tested to set the inputs and record the outputs.

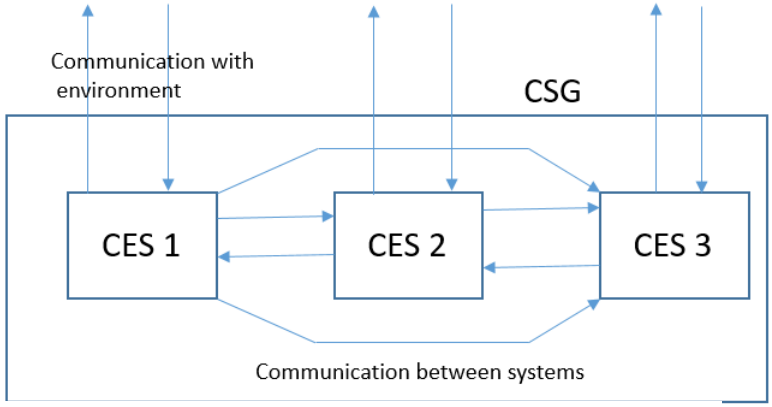


Fig. 5-15: *Communication within a CSG*

Basic approaches for functional testing consider a single embedded system communicating with the environment but not connected to other systems. To test the software of a CES within a CSG, the communication with other systems must be considered. Communication with other systems basically adds new inputs and outputs to the test setup. If, for example, another CES in a collaborative adaptive cruise control sends some information about a traffic jam ahead, this information must be forwarded to the other participants.

Another consideration is to view an entire system group as a single system to be tested. In this case, the communication between several single systems must also be simulated and recorded, just like the communication of a single CES with the environment. Each individual CES communicates with the environment on its own and each CES communicates with other CESs. A schematic representation of this communication of an entire CSG can be seen in Figure 5-15. In our approach, we considered CSGs with a static configuration, which means changes in the reconfiguration such as the addition or removal of CESs are not considered here.

5.5.2 Communication Errors

The collaboration between several CESs adds new challenges to the testing process. An important aspect is to ensure that each individual system is capable of dealing with communication errors. Before we start discussing ways of simulating communication errors, let us introduce two kinds of errors.

www.dbooks.org

In further references, these different kinds of errors will be called detected and undetected errors. If communication errors are detected by the system itself, these errors are called detected errors. In embedded software, detected errors can often be considered as another kind of an “exceptional” input signal. Information such as “communication error occurred” can be considered as “normal” information. Processing that information in a simulation environment is equal to using information like “distance to the car in front.” Typical examples of detected errors are error flags, DTCs (diagnostic trouble codes), and similar data that explicitly signals some malfunction or

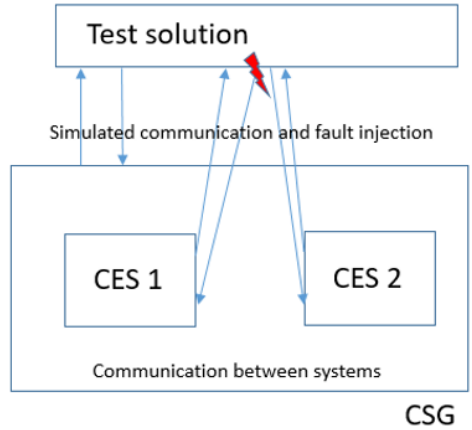


Fig. 5-16: *Communication flow with included test solution*

irregular system behavior. If the detected errors are considered to be a kind of input signal, they can obviously be tested by additionally stimulating those “error detected” flags and recording the behavior in the same way.

On the other hand, undetected communication errors are just the reception or the transmission of incorrect values. To simulate undetected errors during functional safety analysis, the test solution must replace or manipulate the values sent from one CES to another with the desired false values. Following this approach, fault detection mechanisms such as timeout detection of cyclic messages or plausibility checks of input signals can be tested in CSG testing. If the tested system is given incorrect inputs, the behavior of a plausibility check can be verified. By creating the possibility to modify the communication between several collaborative embedded systems, undetected faults can be injected. This approach is called fault injection. It is illustrated in Figure 5-16.

In this figure, instead of sending the information directly from CES 1 to CES 2, the information is sent to the test solution and then forwarded to CES 2. During the modeling of those tests, an additional flag to override certain signal values before forwarding them to CES 2 can be added as part of the test modeling. If this flag is set from the test case modelled, an additionally modeled faulty value can be transferred to CES 2 instead of the actual value from CES 1. The behavior of CES 2, having received the "faulty" value, can still be recorded and evaluated if it fits the specification.

5.6 Conclusion

The development of dynamically coupled collaborative systems poses new challenges for engineers. The methods presented in this chapter support CES engineers in tackling these challenges. They have been selected in order to cover the different design phases and to show that the challenges require continuous consideration of the various aspects along the design process, such as requirements elicitation (including the collaboration of individual CESs in a CSG), functional design that ensures consistency with these requirements, and logical architectures that enable the systems to handle dynamicity, as well as approaches for testing CSG designs.

Some important aspects have been omitted. For example, the design flow introduced in Figure 5-1 shows some "conceptual" flows, which would involve additional methods for the design of intermediate models and analysis results. The aspect of traceability, which would be needed to support engineers in continuously assessing those intermediate design models for adherence to the system requirements, is not discussed either.

5.7 Literature

[Alalfi et al. 2014] M. Alalfi, E. Rapos, A. Stevenson, M. Stephan, T. Dean, J. Cordy: Semi-Automatic Identification and Representation of Subsystem Variability in Simulink Models. In: Intl. Conference on Software Maintenance and Evolution (ICME), 2014.

[Alexander and Maiden 2004] I. Alexander, N. Maiden (Eds.): Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle. Wiley, Chichester, 2004.

[Apel et al. 2013] S. Apel, D. Batory, C. Kästner, G. Saake: Feature-Oriented Software Product Lines: Concepts and Implementation. Berlin/Heidelberg: Springer, 2013.

[Clements et al. 2001] P. Clements, L. Northrop: Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.

www.dbooks.org

[CrESt 2019] CrESt Consortium: EC2.AP2.D2 Methods for Architecture Design of Open Systems, 2019.

[Damm and Harel 2001] W. Damm, D. Harel: LSCs: Breathing Life into Message Sequence Charts. In: Formal Methods in System Design 19:1, 2001.

[Font et al. 2015] J. Font, M. Ballarín, Ø. Haugen, C. Cetina: Automating the Variability Formalization of a Model Family by Means of Common Variability Language. In: Proc. of the Intl. Conference on Software Product Line (SPLC), 2015.

[Franca 2019] Franca project: "Welcome to Franca" November 25, 2019: http://franca.github.io/franca/, accessed on November 25, 2019.

[Harel 1987] D. Harel: Statecharts: A Visual Formalism for Complex Systems. In: Sci. Comput. Programming 8, 1987.

[Harel and Marelly 2003] D. Harel, R. Marelly: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer-Verlag, 2003.

[Marron et al. 2018] A. Marron, Y. Hacohen, D. Harel, A. Mülder, A. Terfloth: Embedding Scenario-Based Modeling in Statecharts. In: MORSE workshop at MoDELS 2018, Copenhagen, 2018.

[Martínez et al. 2014] J. Martínez, T. Ziadi, J. Klein, Y. l. Traon: Identifying and Visualising Commonality and Variability in Model Variants. In: Proc. of the European Conference on Modeling Foundations and Applications (ECMFA), 2014.

[Martínez et al. 2015] J. Martínez, T. Ziadi, T. F. Bissyandé, J. Klein, Y. l. Traon: Automating the Extraction of Model-Based Software Product Lines from Model Variants. In: Proc. of the Intl. Conference on Automated Software Engineering (ASE), 2015.

[Nejati et al. 2007] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, P. Zave: Matching and Merging of Statecharts Specifications. In: Proc. of the Intl. Conference on Software Engineering (ICSE), 2007.

[Nejati et al. 2012] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, P. Zave: Matching and Merging of Variant Feature Specifications. In: IEEE Transactions on Software Engineering (TSE), 2012, pp. 1355-1375.

[Pohl et al. 2005] K. Pohl, G. Böckle, F. van der Linden: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, 2005.

[Pohl et al. 2012] K. Pohl, H. Hönninger, R. Achatz, M. Broy (Eds.): Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology. Springer, Heidelberg/New York, 2012.

[Pohl et al. 2016] K. Pohl, M. Broy, H. Daembkes, H. Hönninger (Eds.): Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES 2020 Methodology. Springer, Heidelberg/New York, 2016.

[Riva and Rosso 2003] C. Riva, C. D. Rosso: Experiences with Software Product Family Evolution. In: Proc. of the Intl. Workshop on Principles of Software Evolution, 2003.

[Rubin and Chechik 2012] J. Rubin, M. Chechik: Combining Related Products into Product Lines. In: Proc. of the Intl. Conference on Fundamental Approaches to Software Engineering (FASE), 2012.

[Rubin and Chechik 2013a] J. Rubin, M. Chechik: N-Way Model Merging. In: Proc. of the European Software Engineering Conference/Foundations of Software Engineering (ESEC/FSE), 2013.

[Rubin and Chechik 2013b] J. Rubin, M. Chechik: Quality of Merge-Refactorings for Product Lines. In: Proc. of the Intl. Conference on Fundamental Approaches to Software Engineering (FASE), 2013.

[Ryssel et al. 2010] U. Ryssel, J. Ploennigs, K. Kabitzsch: Automatic Variation-Point Identification in Function-Block-Based Models. In: Proc. of the Intl. Conference on Generative Programming and Component Engineering (GPCE), 2010.

[Ryssel et al. 2012] J. Ploennigs, K. Kabitzsch, U. Ryssel: Automatic Library Migration for the Generation of Hardware-in-the-Loop Models. In: Science of Computer Programming, 2012, pp. 83-95.

[Schlie et al. 2017] A. Schlie, D. Wille, L. Cleophas, I. Schaefer: Clustering Variation Points in MATLAB/Simulink Models Using Reverse Signal Propagation Analysis. In: Proceedings of the International Conference on Software Reuse (ICSR), 2017. Springer, Salvador, Brazil, 2017.

[Schlie et al. 2018] A. Schlie, S. Schulze, I. Schaefer: Comparing Multiple MATLAB/Simulink Models Using Static Connectivity Matrix Analysis. In: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), 2018. IEEE, Madrid, Spain, 2018.

[Schlie et al. 2019] A. Schlie, C. Seidl, I. Schaefer: Reengineering Variants of MATLAB/Simulink Software Systems. In: Security and Quality in Cyber-Physical Systems Engineering, Springer, 2019.

[Stierand et al. 2016] I. Stierand, P. Reinkemeier, S. Gerwinn, T. Peikenkamp: Computational Analysis of Complex Real-Time Systems - FMTV 2016 Verification Challenge. In: Proc. of the Intl. Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS), 2016.

[Vogelsang 2014] A. Vogelsang: Model-based Requirements Engineering for Multifunctional Systems. Phd thesis, TU München, 2014.

[Wille et al. 2014] D. Wille: Managing Lots of Models: The FaMine Approach, In Proc. of the Intl. Symposium on Foundations of Software Engineering (FSE), ACM, 2014, pp. 817-819.

[Wille et al. 2016] D. Wille, S. Schulze, C. Seidl, I. Schaefer: Custom-Tailored Variability Mining for Block-Based Languages In: Proc. of the Intl. Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016.

[Wille et al. 2018] D. Wille, Ö. Babur, L. Cleophas, C. Seidl, M. v. d. Brand, I. Schaefer: Improving Custom-Tailored Variability Mining Using Outlier and Cluster Detection. In: Science of Computer Programming, no. 163, 2018, pp. 62-84.

[Yakindu 2019] YAKINDU Statechart Tools, November 25, 2019: https://www.itemis.com/en/yakindu/state-machine/, accessed on November 25, 2019.

[Zhang et al. 2011] X. Zhang, Ø. Haugen, B. Møller-Pedersen: Model Comparison to Synthesize a Model-Driven Software Product Line. In: Proc. of the Intl. Software Product Line Conference (SPLC), 2011.

www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Jan Christoph Wehrstedt, Siemens AG
Jennifer Brings, University of Duisburg-Essen
Birte Caesar, Helmut Schmidt University Hamburg
Marian Daun, University of Duisburg-Essen
Linda Feeken, OFFIS e.V
Constantin Hildebrandt, Helmut Schmidt University Hamburg
Wolfram Klein, Siemens AG
Vincent Malik, Siemens AG
Boris Wirtz, OFFIS e.V.
Stefanie Wolf, Siemens AG

6

Modeling and Analyzing Context-Sensitive Changes during Runtime

For collaborative embedded systems, it is essential to consider not only the behavior of each system and the interaction between systems, but also the interaction of systems with their often dynamic and unknown context.

In this chapter, we present a solution approach based on process building blocks—describing both the modelling approach as well as the model execution approach—for engineering and operation to achieve the goal of developing systems that deal with dynamics in their open context at runtime by re-using the models from the engineering phase.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_6

www.dbooks.org

6.1 Introduction and Motivation

CESs operate in open dynamic contexts

Software-intensive embedded systems differ from classic systems in that they interact with their operational context through sensors and actuators [Daun et al. 2016]. The same holds for collaborative embedded systems (CESs) and collaborative (embedded) system groups (CSGs), as their behavior and functions are strongly influenced by changes in the systems' contexts. For example, dynamic traffic conditions—such as pedestrian behavior (which is very difficult to predict), construction work, or other unexpected traffic participants—are major challenges to be met by a platoon of autonomous cars. Similar challenges arise for adaptable and flexible factories in the case of producing individualized products or new variants of the product mix. If the system acts autonomously, a decision has to be taken very quickly on how to react to changes during runtime. This also concerns the operator who has to decide how to deal with the system due to changes.

CESs must be able to cope with context changes during operation

Therefore, methods for coping with these changes during operation must be developed, and this requires the development of suitable methods and models that can be used during operation. These methods must be developed and validated during the engineering phase, which requires the reuse or rather the migration of models from the engineering phase towards runtime.

6.2 Solution Concept

To achieve the goal of developing systems that can handle the dynamics of their open context and to reuse the models from engineering in order to deal with these changes during operation, we present a novel solution approach as described in Figure 6-1; the notation will be explained later in this section.

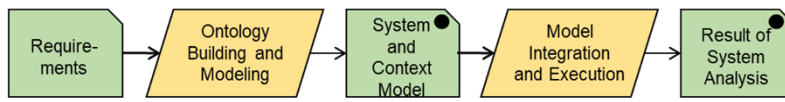


***Fig. 6-1**: Process steps for developing models for CESs interacting with their context and their execution during operation*

Modeling the system and the context

Initially, we develop a modeling approach for both the system and the context: as ontologies are a suitable technology for enabling semantic

interoperability, they allow the creation of information models that link machine-readable meaning to information, thus enabling CESs to mutually understand the shared information. This allows the system and context models to be reused for different applications and also guarantees a certain completeness of the information.

Contributions

Based on that, we provide insights into a scenario analysis approach that identifies, depicts, and analyzes certain contextual situations based on a graphical modeling notation. In this approach, spatial/context constraints are captured as invariants, which change over time. We then present different approaches for model creation, using the example of modeling the capabilities of collaborative embedded systems. Subsequently, in Section 6-4, we describe how to develop decision methods based on these models and runtime information. This leads to the models generated being integrated into executable models that can be used for system analysis. We explain this general approach with different examples: a capabilities check of a system group to fulfill requirements given in a context situation and the seamless integration of simulation for validation of dynamic system behavior.

CrESt process building block notation

Some of the methods presented can be broken down into a set of sub-methods with certain interdependencies and specific types of artifacts (see Figure 6-2). To show the relationships between such sub-methods (e.g., how to combine them), to clearly assign them to the design phase or to the runtime of a CES/CSG, and to classify their degree of formalization and automation, we choose the following notation:

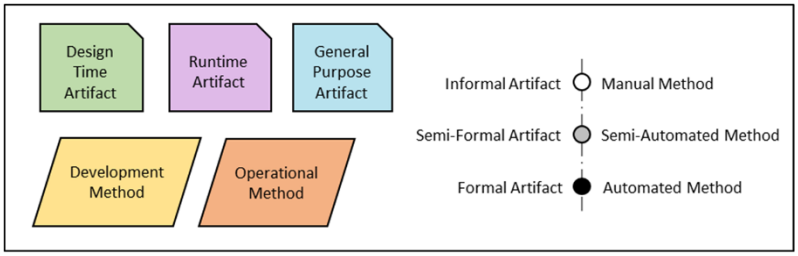


Fig. 6-2: *CrESt process building block (PBB) notation*

6.3 Ontology and Modeling

CESs operate in open contexts

The development and operation of CESs that can cope with a frequently changing environment has become a major field of research. Of particular interest is a CES's ability to operate in open contexts, that is, situations where context objects (e.g., other CESs)

www.dbooks.org

enter or leave the CES's context at runtime [Schlingloff et al. 2016]. As CESs are usually embedded in a network of CESs (i.e., a CSG), an individual CES must be able to process and communicate complex information from/to changing communication partners during its life cycle in order to provide its functionality.

Ontologies as a solution concept

This addresses the development of the models as well as the modeling approach itself. Ontologies are becoming the appropriate means for these systematic approaches. In the following section, we therefore address a modeling approach for the system and context to develop models that can cope with online decisions.

6.3.1 Ontology Building

Ontologies must be derived systematically

Ontologies provide a suitable technology for formalizing different aspects of a CES and could potentially be the provider of data, information, and knowledge for different software functionalities [Sabou et al. 2019]. In the case of a CES in the manufacturing domain, an ontology can be used to formalize manufacturing-related capabilities (see Sections 6.3.2 and 6.4.2), features relevant for reconfiguration of a manufacturing system (see Section 6.3.3), or serve as input information for a simulation of the manufacturing system (see Section 6.4.1). The development of an ontology is a non-trivial task that requires high efforts from different stakeholders. Therefore, an efficient ontology building procedure is crucial in order to have a positive cost-benefit trade-off. The latter is the goal of the method described below, which is summarized by Figure 6-3.

Ontologies define terminologies

An ontology may take a variety of forms, but it will of necessity include a vocabulary of concepts and some specification of their meaning, including definitions and an indication of how concepts are interrelated. In general, the above-mentioned concepts and relationships (including attributes) form what is referred to as the terminology box (TBox), while the axioms (e.g., individuals of the ontology) form the assertional box (ABox) that follows the definitions of the TBox [Hildebrandt et al. 2018].

A three-step approach to ontology building

The method consists of three PBBs. The first PBB focuses on the elicitation of requirements for the ontology under development and the documentation of these requirements. The elicitation of ontological requirements, as presented in [Hildebrandt et al. 2018a], begins with a set of project requirements. During the documentation of the requirements of a CES, requirement models (e.g., sequence charts) of the CES that uses the ontology are annotated with ontological requirements. Ontological requirements are stated as

competency questions. These are an informal notion of a query that should be answered by the ontology in a certain way, for example, “Which processes can be performed by machine X and what sensor measures which data?”.

The second PBB is concerned with building the TBox of the ontology under development, which is described in detail in

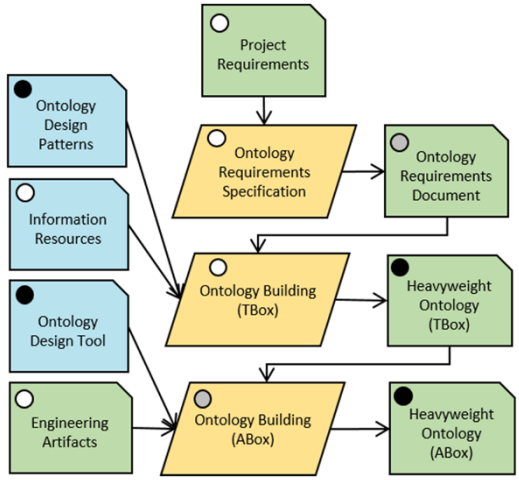


Fig. 6-3: *Process building block for the ontology building method*

[Hildebrandt et al.2020]. This PBB begins with a search for information resources that provide the relevant terms and relationships in order to build the ontology under development. Potentially suitable information resources are industry standards (e.g., ISO or IEEE standards [VDI 2005] for the process description or [IEC 2016] for properties and features), scientific publications, or project reports. Standards are preferred due to the high maturity of the concepts and relationships used as well as their potentially wide dissemination. When the proposed method is applied, heavyweight ontologies are created which formalize the knowledge of a domain (e.g., process description) in a reusable manner and can therefore be used as ontology design patterns. As presented in [Hildebrandt et al. 2020] several ontology design patterns have already been published. In an ontology building project, these ontology design patterns are combined to obtain the ontology under development.

Building the ABox

After having built the TBox of the ontology, we can begin with building the ABox of the ontology, which is sometimes referred to as the knowledge graph. The ABox contains all the relevant real-world facts (e.g., features or properties of a manufacturing process “Milling”

www.dbooks.org

that was or will be performed) that should be contained in the answers to the competency questions stated at the beginning. In order to create the ABox, we use the ontology design tool we developed, which automatically transforms engineering artifacts (e.g., 3D CAD, AutomationML [Lüder et al. 2017]) into the ABox via an extract, transform, load pipeline. If there are no engineering artifacts available, we use a semi-automatic approach that relies on ontology design patterns again, see [Hildebrandt 2020a].

The final result is an ontology that describes facts about a CES that can be shared with other CESs. An application of an ontology in the domain of CESs is shown in Section 6.3.3, 6.3.3, 6.4.1, and 6.4.2 respectively.

6.3.2 Capability Modeling

Capabilities of technical systems

The capability of a technical system describes whether and with what quality the system is able to perform a specific task or fulfill a specific goal, while the task or the goal may vary with the context the system is operating in. The quality of the activity that is performed by the system depends on the context (i.e., on current requirements or boundary conditions), as well as on the immanent properties of the system (cf. [Reschka 2016], [Weiß et al. 2019]).

Capability models formalize CES and CSG capabilities

Capability models are used to formalize and document the capabilities of a CES or CSG. They are usually defined at design time of a CES or CSG or even prior to that, but they are predominantly used for runtime evaluations (e.g., a manufacturability check of a product based on the capabilities of available production systems).

Approach for creating capability models

Figure 6-4 gives an overview of the creation of capability models, distinguishing between the metamodel level, the domain capability model, the potential extension of the domain model based on project-specific boundary conditions, and the integration of the project-specific capabilities into the system model.

In the following, we explain the four sub-methods of the capability modeling approach, shown in Figure 6-4, in more detail.

Capability metamodel creation: The capability metamodel defines the structure of a capability model and its abstract syntax (cf. [Sprinkle et al. 2010]). It is neutral to any domain or application case. The metamodel is created based on a specification that may be derived from the requirements that various domains have with

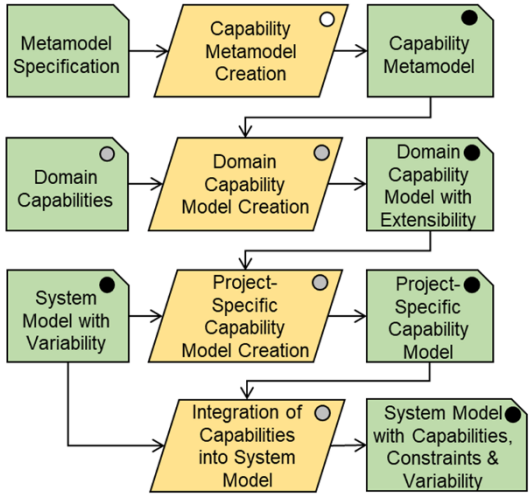


Fig. 6-4: *Method for creating capability models*

respect to capability descriptions. We present a basic version of such a metamodel for capabilities in Figure 6-5.

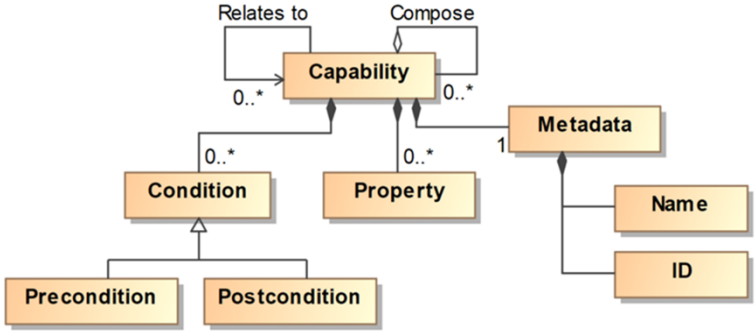


Fig. 6-5: *Metamodel for capabilities*

In this metamodel, a capability is characterized by metadata such as a name and an ID, as well as by properties and pre- or postconditions (e.g., a required condition of a workpiece before applying a production capability in the discrete manufacturing domain). In addition, capabilities may relate to each other or may in turn be composed of capabilities. Such characteristics of a capability can be standardized for a certain domain by developing domain capability models.

www.dbooks.org

Domain capability model

Domain capability model creation: The capability model for a certain domain is created by identifying the capabilities of a domain based on its defined scope and by considering existing ontologies as well as expert knowledge and industry standards. These domain capabilities are the input for a domain capability model that is independent from any specific system manufacturer. In addition, the domain capability model should be extendable in order to enable project-specific modifications (e.g., for niche or special applications or in case of technical innovations). To give an example, a model for manufacturing capabilities that applies to adaptable and flexible factories in the discrete manufacturing domain can be created. A domain capability model for discrete manufacturing must cover a defined structure and uniform nomenclature, as well as a means of describing manufacturable product features and typical restrictions and boundary conditions of the production systems. Moreover, potential interdependencies between capabilities—for example, with regard to production process chains and assembly sequences—must be taken into account (cf. [Wolf et al. 2020]). Example 1-1 shows an exemplary capability description in the discrete manufacturing domain:

Example 6-6: Capability "drilling"

Metadata:

Name: Drilling

ID: 331-01

Properties (*excerpts only*):

Related manufacturing features: blind hole, through-hole

Diameter (mm)

Depth (mm)

Project-specific capability model

Project-specific capability model creation: For niche or special applications, or in the case of technical innovations, extensions to the existing domain capability model may be necessary. When a project-specific capability model is created, the domain capability model is used as a basis. The system under development, for which the capabilities need to be modeled, provides the input for the extension of the domain capability model. The result of this method is a project-specific capability model applicable to CESs and CSGs.

Integration into system model

Integration of capabilities into the system model: Finally, within a specific project, the capabilities of each CES and/or CSG must be described and interlinked with the system model. In this step, capabilities are assigned to the system and its sub-systems while

taking the system-specific constraints on these capabilities into account. Example 6-7 shows the capability "drilling" of an exemplary machine with specific values. Please note that the generic capabilities modeled in the domain capability model can be instantiated not only for a specific production setup (i.e., distinct machines), but also for specific products to be produced to allow manufacturability checks during the operation phase of a factory (see Section 6.4.2 "Capability Matching" for further details). With regard to a manufacturing system, the properties of a capability may have a range of possible values (cf. Example 6-7), whereas in the case of a product to be produced, properties may have just a single value (e.g., diameter d = 10 mm).

Example 6-7: Capability "drilling" of Machine A

Metadata:
- Name: Drilling
- ID: 331-01

Properties (*excerpts only*):
- Related manufacturing features: blind hole, through-hole
- Diameter (mm): 3 - 30
- Depth (mm): <= 100

Capabilities are subject to variability

As the capabilities of a system may be subject to variability, care must be taken to ensure that a differentiation between current and theoretical capabilities is possible. This is especially relevant if a system performs a reconfiguration or re-parameterization, as this usually implies changes in the capabilities of the system (see Section 6.3.3. The artifact that is finally generated is the system model, with capabilities and variability, which forms the basis for runtime evaluations.

Capability models can be implemented as ontologies (e.g., using the methodology in Section 6.3.1) or as feature models (cf. Section 6.3.3 and [Wolf et al. 2020]).

6.3.3 Variability Modeling for Context-Sensitive Reconfiguration

Reconfiguration during runtime extends the available capability

As stated in Section 6.3.2, certain CESs can be reconfigured during runtime, which means that the capabilities provided in a certain state can differ from the capabilities in a future state. This ability to change between different states is called reconfiguration. As CESs interact in an open and dynamic environment, a context-sensitive reconfiguration is desirable. Therefore, the goal of the engineering

www.dbooks.org

method variability modeling for context-sensitive reconfiguration is to support the creation of context-sensitive variability models (CSVM) for runtime usage. Context-sensitive variability models are dedicated models that represent different configurations a system can take. According to [Mauro et al. 2016], the problem space of a context-sensitive variability model consists of three parts: a feature model, a context model, and cross-tree constraints. Accordingly, the engineering method must include a separate creation phase for each part.

Figure 6-8 shows the overview of the engineering method, as well as the runtime usage. The first step is the creation of a feature model

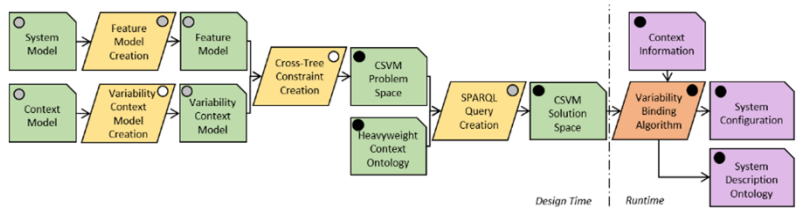


Fig. 6-8: *Procedural overview variability modeling for context-sensitive reconfiguration*

in which all common and variable parts of a system are captured [Kang et al. 1990]. To identify and extract this information, the system model, which is composed of different engineering artifacts, must be analyzed. For a manufacturing system, this system model could be a 3D-CAD drawing in the form of a step file or control code according to IEC 61131. In the second step, a variability context model is generated that contains all relevant context information for triggering the reconfiguration of a system. For this purpose, the system's context must be analyzed. Accordingly, a concept is developed that is illustrated for the manufacturing domain and helps to identify the relevant information for reconfiguration — for example, other CESs providing certain capability such as handling or a certain product requirement such as a drilling hole diameter. For this purpose, different approaches, for example, [Marks et al. 2018], are analyzed and combined. Subsequently, to conclude the third step of the engineering procedure, the context model must be integrated with the feature model by formulating cross-tree constraints.

Cross-tree constraints are used to describe the dependencies between components that are required, for example, to provide a certain capability or to specify that a certain capability can only be provided in a certain configuration. These logical formulas are phrased as

described in [Kang et al. 1990]. Once the third step of the engineering procedure has been completed, the problem space of the context-sensitive variability model is defined. The solution space is then used to enable the system to provide a self-description of its current configuration, including a description of the capabilities available. Therefore, the fourth step of the engineering method is the SPARQL query creation. SPARQL [SPARQL 2020] is a query language that builds upon the W3C standard Web Ontology Language [OWL 2020] and can be used to create, update, or query ontologies.

To create the SPARQL statements, the terminology box (TBox) that comprises the terms and relationships for describing a real-world phenomenon in an abstract manner must be considered [Asunción et al. 2004]. In the case of reconfiguration of modular manufacturing systems, the TBox describes how each system has to be characterized to be able to collaborate with other modules of the manufacturing system — for example, the module type package in the process industry [Ladiges et al. 2018], referred to in Figure 6-8 as "Heavyweight System Ontology," which is created following the ontology building method of Section 6.3.1. Thus, each reconfiguration requires an update of the system description such that it is always aligned with the current configuration of the manufacturing system. The SPARQL statements created are used to create and alter the assertional box (ABox), which contains the axioms (i.e., individuals of the ontology) [Baader et al. 2003] that represent the system description. The SPARQL statements are separated into snippets and related to features of the feature model. Only those features are represented in the system description that are selected in the current configuration. A detailed description can be found in [Caesar et al. 2019]. Once the fourth step is complete, the problem and solution space of the context-sensitive variability model is defined and can be used for reconfiguration during runtime, see Figure 6-8. Details of the variability binding algorithm can be found in Chapter 18.

6.3.4 Scenario-Based Modeling

Scenarios can be described by use case models

During requirement elicitation, use case descriptions are a well-established means of gaining insight into the system to be designed. However, use case descriptions and requirement models (as needed in Section 6.3.1 for ontology building) are often informal and lack a concise semantics, meaning that it is difficult to reuse them later on in the development process. If scenarios are expressed using a specification language that is intuitive and lightweight but still concise

www.dbooks.org

and with a formal background, integration in the overall process (and even scenario-based development) becomes much easier.

Traffic sequence charts visualize traffic scenarios

In traffic-related applications (such as the implementation of maneuvers of a platoon of vehicles on a highway), it is especially important to express spatial properties and constraints. Traffic Sequence Charts (TSCs) are a visual formalism that allows intuitive and concise specification of traffic scenarios based on a formal semantics [Damm et al. 2018]. TSCs are based on acyclic graphs of chart nodes. Chart nodes capture constraints over a time interval and can be combined into a chart using sequence, choice, or parallel composition. In a chart, the constrained time intervals are seamless — that is, there is no time gap within a sequence of two nodes. Chart nodes include simple invariants (constraints that hold throughout the complete constrained interval), conditions (constraints holding at least once), and complex nodes specifying communication/event patterns or containing complete charts.

Spatial views describe spatial constraints between objects, represented by symbols in a topological view. Spatial views can be used as base constraints for invariant and condition nodes. TSCs are interpreted with respect to a modular world model that defines object classes, interfaces, and (if necessary) behaviors. World model modules are exchangeable, as long as appropriate interfaces are provided.

Spatial views are used as base constraints

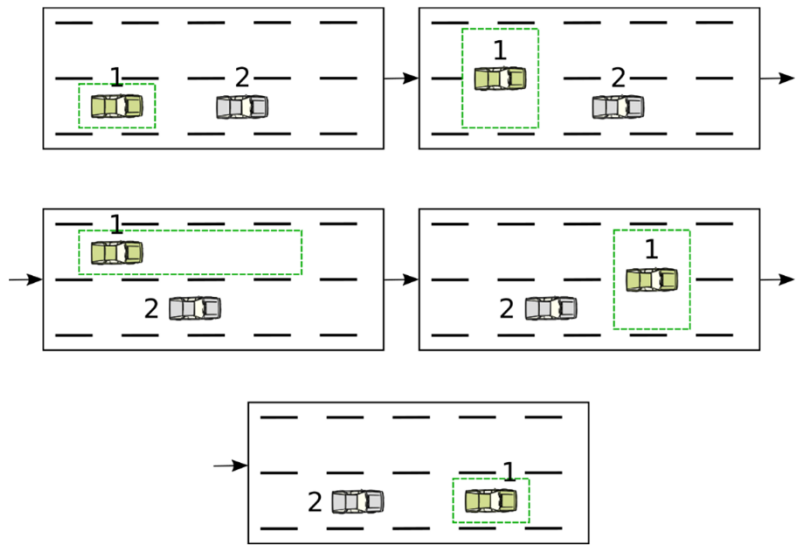


Fig. 6-9: *TSC excerpt of an overtaking maneuver*

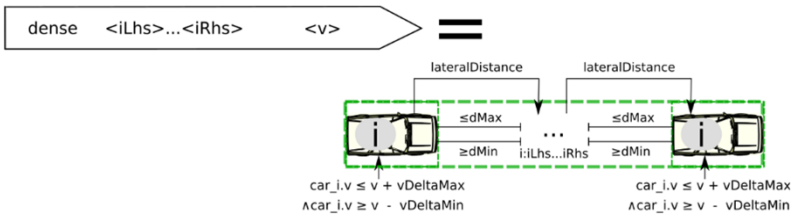


Fig. 6-10: *Vehicle group symbol abbreviation*

The combination of spatial, time, and communication constraints allows traffic scenarios to be defined intuitively. For example, the TSC excerpt in Figure 6-9 describes an overtaking maneuver: during the first invariant, car 1 is somewhere (within the box with dotted lines) behind car 2. During the second invariant, car 1 is still behind car 2, but somewhere in both lanes. In the third invariant, car 1 is somewhere next to car 2, etc.

Spatial patterns

An important aspect for a specification language for CSGs is to represent spatial patterns regardless of the concrete number of systems in the group. For example, a platoon driving on a highway lane can be considered as a sequence of vehicles on that lane connected by spatial and non-spatial relationships. In a TSC, this is expressed using the ellipsis notation; an example can be seen in the lower part of Figure 6-10. The individual vehicles are represented by car symbols parameterized with the position of the vehicle in the group, ranging from iRhs (head of the group) to iLhs (tail of the group). All vehicles drive in the same velocity range, and each adjacent pair of vehicles travels in bounded longitudinal and lateral distance.

Abstractions help manage complexity

Showing detailed information for many objects can quickly become overwhelming and leads to important information being overlooked and lost. Therefore, TSCs offer the possibility to define on-the-fly visual abstractions by introducing abbreviation symbols. These symbols abbreviate spatial patterns in full depth but can be customized to display the most important information only, therefore (in some sense) highlighting the relevant essence of the group in the current situation. For example, Figure 6-10 introduces an abbreviation symbol that is parameterized in a start and end vehicle index (thus allowing selection of subgroups) as well as traveling speed. The individual vehicles are represented by car symbols.

Figure 6-11 shows two invariants being valid for the same time interval. The upper invariant describes the inner structure of a platoon, the lower one its relationship to the context. As the picture describes normal operation, there must be no other vehicle in the

www.dbooks.org

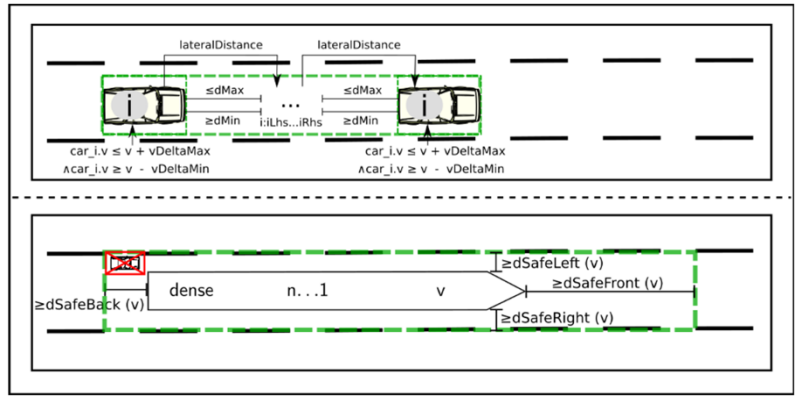


Fig. 6-11: *Separation of concerns: platoon inner structure and context*

immediate neighborhood of the platoon (depicted by the small crossed out car in the upper left corner of the box with dotted lines).

Patterns allow specification of complex situations

The patterns introduced allow the specification of complex collaborative maneuvers, such as how a platoon can circumvent an obstacle. The integrated formalism of TSCs provides the possibility to analyze the specified maneuvers, for example, by means of consistency checks as presented in [Becker 2020].

6.4 Model Integration and Execution

Operational behavior must be validated

During operation, the behavior of the system or the CSG must be validated taking the context into consideration. This can be done using simulation models, for example. Because generating these models requires high effort, it is unavoidable to automatically generate and calibrate them based on data from the real system and the knowledge of the system and context behavior. This leads to an integration of simulation into the design and runtime phase.

6.4.1 Model Generation for Simulation Models

Model Generation via Knowledge Graph

Coping with heterogeneous input data

With heterogeneous input data for the system and context, the automated generation of an executable model requires knowledge of the given system and context structure as well as an efficient connection of input data. As the context and system information are usually stored in heterogeneous data sources and formats, a common

data model, including context as well as system information, must be extracted and combined.

Ontologies help with managing the variety of data

This variety of data can be managed using ontologies as described above, represented through the application of a knowledge graph (see Section 6.3.1). In order to gain a single source of data description, the data model of the system, as well as the context used in the knowledge graph, is linked to the original data sources and formats. Queries can then be used to filter the relevant data and their linkage for the generation of simulation models. The models are matched using defined interfaces. As output of the knowledge graph, both models are extracted by export functions using requests leading to an executable simulation model.

The following properties of the knowledge graph make this approach applicable as an interface for heterogenous input data in dynamic context: in order to avoid the high effort of the classical and often manual generation of models as described above, the knowledge graph approach requires no predefined data model, offers fast access to complex hierarchical structures, as well as semantic search and analysis.

Application to a Real Production System

During operation of a production system, there are several ways to operate the system depending on requirements regarding available resources, production orders, or production time and costs. Therefore, the future system behavior must be predicted and the best operation strategy identified based on the current state.

The generalized system and context data models in the knowledge graph are concretely filled with current context information such as production plans, resource availability, and product mix provided by the *context information* artifact [Rosen et al.2020]. In addition, the system model is represented by different engineering artifacts, such as operation strategies, the production machines and their capabilities (see Section 6.3.2), as well as the plant layout (see Figure 6-12).

Simulation can be used for different tasks

Depending on the simulation task—for example, material flow analysis or 3D-kinematic analysis—the adapted data from the knowledge graph can be used to generate different types of simulation models for discrete or continuous production processes. For both aspects, the independent generation of the context and system model represented either as a process model or as operation strategies, including their coupling, is possible. Moreover, different *simulation*

www.dbooks.org

models can be generated automatically by varying the parameters

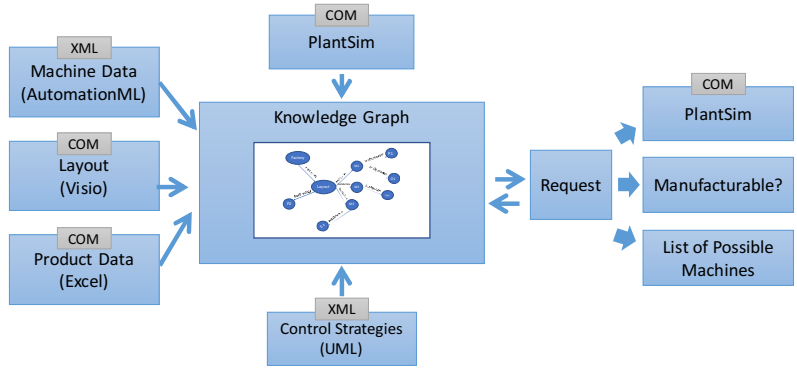


***Fig. 6-12**: Knowledge graph for factory simulation*

dependent on the operation strategy. A final assessment of the different simulation results is performed using selected KPIs (e.g., throughput, buffer utilization, productivity, costs, energy) in order to identify the "best" operation strategy.

For example, as described in Section 6.3.3 for different configurations, simulation models can be generated to select the best suitable reconfiguration or rather to provide the evaluation of different strategies to allow the operator to make a decision (see Chapter 3).

6.4.2 Capability Matching

Required capabilities may change due to context dynamicity

Context dynamicity leads to rapid changes in the operational environment of a CES (cf. [Tenbergen et al. 2018]). To cope with these changes, CESs may dynamically recombine at runtime to form a CSG that aims to fulfill a certain, usually context-dependent, goal. Note that CESs and CSGs do not always share the same goals or aim to fulfill complementing goals [Daun et al. 2019]. Due to the dynamic formation, configurations can occur where individual participants aim to achieve conflicting goals [Brings 2020]. As the dynamic formation of CSGs at runtime can hardly be foreseen at design time, a method is needed to examine whether a system group configuration actually provides the capabilities that are required under certain contextual boundary conditions. Therefore, we developed a method, based on model matching techniques, that enables such examinations by applying the following step-by-step approach (cf. Figure 6-13):

Capability matching procedure

(1) Derivation of required capabilities: The first step is to determine which generic capabilities and especially which combinations of

capabilities meet the requirements imposed by the goal or task of the CSG (e.g., the fulfillment of a customer's production request in an adaptable and flexible factory). The domain capability model introduced in Section 6.3.2 can be used to describe these capabilities needed for a certain goal attainment.

(2) *Matching of available and required capabilities and determination of suitable CSG configurations:* The next step is to answer the question of whether, based on the available CESs, a CSG can be formed that is able to provide these required capabilities. At this point, the system models of the individual CESs, with their capabilities and corresponding variability, are mapped to the required capabilities in order to identify suitable capability combinations and determine appropriate CES and CSG configurations (cf. Section 6.3.3 for reconfigurations).

(3) *Evaluation of alternatives*: Finally, if there is more than one possibility to form the CSG, the most appropriate option must be identified by using optimization criteria or considering timing or strategic aspects. The results are a certain combination of capabilities with allocated systems and a defined CSG configuration.

Figure 6-13 shows the process building blocks for this method.

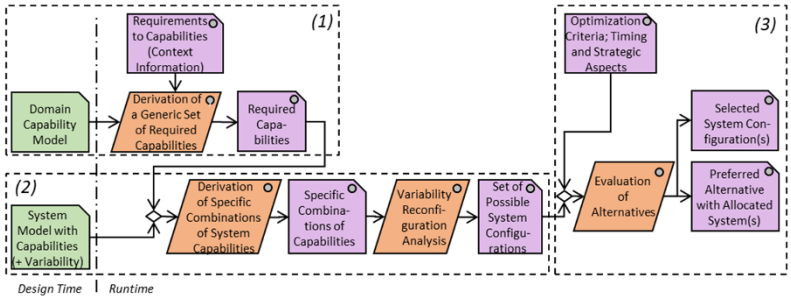


Fig. 6-13: *Process building blocks for capability matching*

Capability matching for the adaptable and flexible factory

In the following, we illustrate the application of this method for the adaptable and flexible factory use case. In this use case, the requirements for the capabilities of CESs arise from a production request for an individualized product. The factory is equipped with production systems represented by various CESs and we must examine whether they can form a suitable CSG for the fulfillment of the production request.

www.dbooks.org

Instantiation of generic capabilities from the domain capability model

Figure 6-14 illustrates schematically how the generic capabilities of the domain (or project-specific) capability model (see Section 6.3.2) are instantiated for the product and the production systems, thus generating the required and provided capabilities that form the basis for the matching.

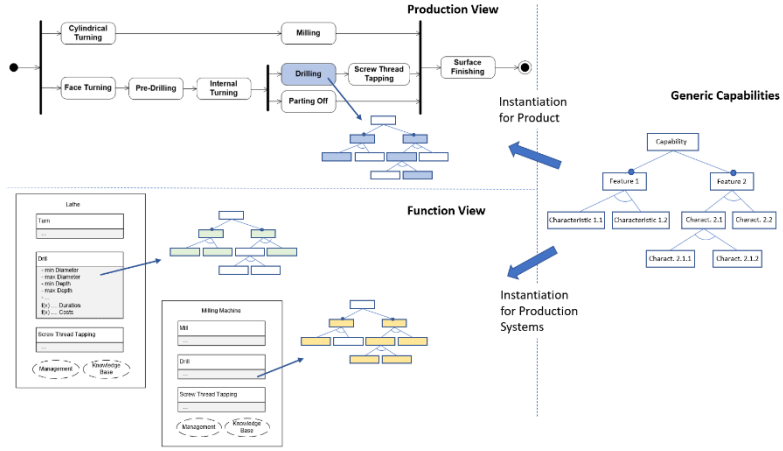


Fig. 6-14: *Schematic sketch showing the instantiation of generic capabilities*

For capability matching, a check determines whether the required capabilities, represented by the production view in Figure 6-14, match the capabilities provided by the resources of the factory as shown in the function view in Figure 6-14.

Integration of production and function view

Consequently, the integration of the production view and the function view allows us to examine whether a certain product can be produced by the adaptable and flexible factory. Figure 6-15 illustrates the combination of the production view and the function view.

The figure shows three machines—lathe, milling machine, and polishing machine—as well as their production functions (i.e., instantiated capabilities from the domain or project-specific capability model). In addition, two different production process sequence variants for manufacturing the product with the given production systems are shown. There are some common steps between these two production processes (e.g., at the beginning, the raw material is first turned with the lathe, then drilled and turned again). Other steps differ: for example, screw thread tapping is conducted either on the lathe (Example I) or on the milling machine (Example II). In both cases, different intermediate products are exchanged between the lathe and the milling machine. Depending on the choice made, the

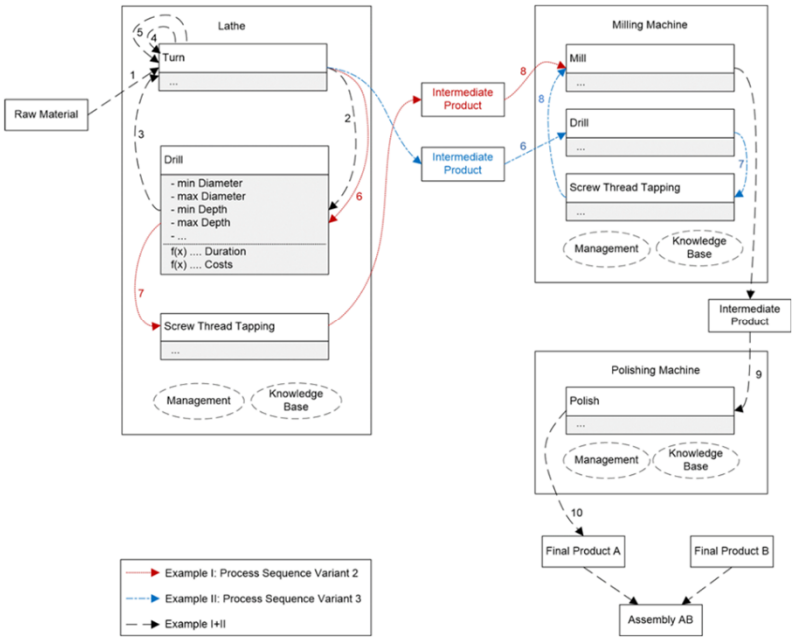


Fig. 6-15: *Integration of production view and function view to check manufacturability*

process can differ in time and costs. Therefore, the time and costs for each step must be calculated so that the optimal solution can be found.

For further information on the views presented and the principles of the matching method, please refer to [Daun et al. 2019a].

6.5 Conclusion

This chapter illustrated a modeling approach for analyzing the behavior of CESs during operation by re-using models from the engineering phase. We illustrated this approach for selected examples, addressing the main line of this developing approach. To improve the quality and reduce the effort for each step, additional improvements are necessary that lead to reusable ontologies, standardization of concepts, and interfacing to allow integration of tools. This leads to possible extensions not described in this chapter.

Even if the model generation process can then be executed automatically, a lot of effort is still required to develop the underlying ontologies in advance. Therefore, the ABox and TBox necessary for building the ontologies based on existing and established engineering artifacts must also be developed. Using databases also reduces this

www.dbooks.org

effort as the manual mapping between the ABox, TBox, and the industrial application can be reused more easily [Hildebrandt 2020a].

In addition to further reduction of efforts for the modeling, automatic model validation is also a big benefit. This can be done using review models [Daun et al. 2020]. All these approaches are part of a vision to introduce model-based development approaches to non-experts in engineering and operation and efficient model generation and execution during runtime.

6.6 Literature

[Asunción et al. 2004] G.-P. Asunción, F.-L. Mariano, O. Corcho: Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web. 1st Edition, Springer-Verlag, London, 2004.

[Baader et al. 2003] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, D. Nardi: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge university press, 2003.

[Becker 2020] J. S. Becker: Partial Consistency for Requirement Engineering with Traffic Sequence Charts. Software Engineering, 2020.

[Brings 2020] J. Brings, M. Daun, T. Weyer, K. Pohl: Goal-Based Configuration Analysis for Networks of Collaborative Cyber-Physical Systems. In: 35th ACM Symp. Applied Computing, 2020, pp. 1387–1396.

[Caesar et al. 2019] B. Caesar, M. Nieke, A. Köcher, C. Hildebrandt, C. Seidl, A. Fay, I. Schaefer: Context-Sensitive Reconfiguration of Collaborative Manufacturing Systems. In: 9th IFAC Conf. Manufacturing Modelling, Management and Control (MIM), Germany, Berlin, 2019, pp. 307–312.

[Damm et al. 2018] W. Damm, E. Möhlmann, T. Peikenkamp, A. Rakow: A: Formal Semantics for Traffic Sequence Charts. In: Principles of Modeling – Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday, 2018, pp. 182-205.

[Daun et al. 2016] M. Daun, B. Tenbergen, J. Brings, T. Weyer: SPES XT Context Modeling Framework. In: Advanced Model-Based Engineering of Embedded Systems, Springer, 2016, pp. 43-57.

[Daun et al. 2019] M. Daun, V. Stenkova, L. Krajinski, J. Brings, T. Bandyszak, T. Weyer: Goal Modeling for Collaborative Groups of Cyber-Physical Systems with GRL. In: 34th ACM Symp. Applied Computing, 2019, pp. 1600–1609.

[Daun et al. 2019a] M. Daun, J. Brings, P.A. Obe, et al.: Using View-Based Architecture Descriptions to Aid in Automated Runtime Planning for a Smart Factory. In: 2019 IEEE Int. Conf. Software Architecture (ICSA-C). pp. 202–209.

[Daun et al. 2020] M. Daun, J. Brings, T. Weyer: Do Instance-Level Review Diagrams Support Validation Processes of Cyber-Physical System Specifications: Results from a Controlled Experiment. In: Int. Conf. Software and System Processes, 2020.

[Hildebrandt 2020a] C. Hildebrandt: Lightweight Industrial Ontology Design Support: A Tool for ABox Creation Based on Ontology-Design Patterns. Online: https://github.com/ConstantinHildebrandt/lion; accessed on: 02/28/2020.

[Hildebrandt et al. 2018] C. Hildebrandt, S. Törsleff, B. Caesar, A. Fay: Ontology Building for Cyber-Physical Systems: A Domain Expert Centric Approach. In: 14th IEEE Conf. Automation Science and Engineering, CASE, 2018, pp. 1079-1086.

[Hildebrandt et al. 2018a] C. Hildebrandt, S. Törsleff, T. Bandyszak, B. Caesar, A. Ludewig, A. Fay: Ontology Engineering for Collaborative Embedded Systems – Requirements and Initial Approach. In: Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018, pp. 57-66.

[Hildebrandt et al. 2020] C. Hildebrandt, A. Köcher, C. Küstner, C.-M. Lopez-Enriquez, A.W. Müller, B. Caesar, C.S. Gundlach, A. Fay: Ontology Building for Cyber-Physical Systems: Application in the Manufacturing Domain. *IEEE T. Autom. Sci. Eng.*, 2020.

[IEC 2016] International Electrotechnical Commission (IEC). (2016) IEC 61360 - Common Data Dictionary (CDD - V2.0014.0014). Online: http://cdd.iec.ch/cdd/iec61360/iec61360.nsf, accessed on: 02/28/2020.

[Kang et al. 1990] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Carnegie-Mellon University, USA, Pennsylvania, Pittsburgh, 1990. – Technical Report.

[Ladiges et al. 2018] J. Ladiges, A. Fay, T. Holm, U. Hempen, L. Urbas, M. Obst, T. Albers: Integration of Modular Process Units Into Process Control Systems. In: IEEE Transactions on Industry Applications, Vol. 54, No. 2, 2018, pp. 1870–1880.

[Lüder et al. 2017] A. Lüder, N. Schmidt: AutomationML in a Nutshell. In: Handbuch Industrie 4.0 Bd. 2. Springer Vieweg, Berlin, Heidelberg, 2017, pp. 213-258 (available in German only).

[Marks et al. 2018] P. Marks, X. L. Hoang, M. Weyrich, A. Fay: A Systematic Approach for Supporting the Adaptation Process of Discrete Manufacturing Machines. In: Research in Engineering Design Vol. 29, No. 4, 2018, pp. 621–641.

[Mauro et al. 2016] J. Mauro, M. Nieke, C. Seidl, I. C. Yu: Context-Aware Reconfiguration in Evolving Software Product Lines. In: Tenth Int. Workshop on Variability Modelling of Software Intensive Systems, USA, New York, 2016, pp. 41–48

[OWL 2020] OWL 2 Web Ontology Language – World Wide Web Consortium (W3C), https://www.w3.org/TR/owl2-overview/; accessed on 04/27/2020.

[Reschka 2016] A. C. Reschka: Fertigkeiten- und Fähigkeitengraphen als Grundlage des sicheren Betriebs von automatisierten Fahrzeugen im öffentlichen Straßenverkehr in städtischer Umgebung. Dissertation, Technische Universität Braunschweig, 2017 (available in German only).

[Rosen et al. 2020] R. Rosen, D. Beyer, J. Ficher, W. Klein, V. Malik, J.C. Wehrstedt: Flexible Produktion durch digitale Zwillinge in der Produktion. In: Kongress Automation 2020, VDI, 2020 (available in German only).

[Sabou et al. 2019] M. Sabou, S. Biffl, A. Einfalt, L. Krammer, W. Kastner, F.J. Ekaputra: Semantics for Cyber-Physical Systems: A Cross-Domain Perspective. In: Semantic Web – Interoperability, Usability, Applicability, 2019.

[Schlingloff et al. 2016] B.-H. Schlingloff, H. Stubert, W. Jamroga: Collaborative Embedded Systems — A Case Study. In: 3rd Int. Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), 2016, pp. 17–22.

[SPARQL 2020] SPARQL Query Language for RDF – World Wide Web Consortium (W3C https://www.w3.org/TR/rdf-sparql-query/ accessed on 07/23/2020.

www.dbooks.org

[Sprinkle et al. 2010] J. Sprinkle, B. Rumpe, H. Vangheluwe, G. Karsai: Metamodelling: State of the Art and Research Challenges. In: Model-Based Engineering of Embedded Real-Time Systems. Int. Dagstuhl Workshop, 2007, Springer Berlin Heidelberg 2010, pp. 57-76.

[Tenbergen et al. 2018] B. Tenbergen, M. Daun, P. A. Obe, J. Brings: View-Centric Context Modeling to Foster the Engineering of Cyber-Physical System Networks. In: *IEEE Int. Conf. Software Architecture, ICSA,* 2018, pp. 206–216.

[VDI 2005] VDI/VDE Richtlinie 3682: Formalised process descriptions - Concept and graphic representation, Beuth Verlag Berlin 2005-9.

[Weiß et al. 2019] S. Weiß, B. Böhm, J. Vollmar, B. Caesar, A. Fay: Modellierung von Fähigkeiten industrieller Anlagen für die auftragsgesteuerte Produktion. In: Kongress Automation 2019, VDI, 2019 (available in German only).

[Wolf et al. 2020] S. Wolf, B. Caesar, A. Fay, B. Böhm: Erstellung eines Domänenmodells zur Beschreibung von Fähigkeiten fertigungstechnischer Anlagen für die auftragsgesteuerte Produktion. In: Kongress Automation 2020, VDI, 2020 (available in German only).

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Torsten Bandyszak, University of Duisburg-Essen
Lisa Jöckel, Fraunhofer IESE
Michael Kläs, Fraunhofer IESE
Sebastian Törsleff, Helmut Schmidt University Hamburg
Thorsten Weyer, University of Duisburg-Essen
Boris Wirtz, OFFIS e.V.

7

Handling Uncertainty in Collaborative Embedded Systems Engineering

As collaborative embedded systems operate autonomously in highly dynamic contexts, they must be able to handle uncertainties that can occur during operation. On the one hand, they must be able to handle uncertainties due to the imprecision of sensors and the behavior of data-driven components for perceiving and interpreting the context to enable decisions to be made during operation. On the other hand, uncertainties can emerge from the collaboration in a collaborative group, related to the exchange of information (e.g., context knowledge) between collaborative systems. This chapter presents methods for modeling uncertainty early in development and analyzing uncertainty during both design and operation. These methods allow for the identification of epistemic uncertainties that can occur when various, potentially heterogeneous systems are required to collaborate. The methods also enable graphical and formal modeling of uncertainties and their impact on system behavior (e.g., in the course of dynamic traffic scenarios). Furthermore, this chapter investigates the quality of outputs issued by data-driven models used to equip collaborative embedded systems with uncertainty-resilient machine learning capability.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_7

www.dbooks.org

7.1 Uncertainty in Collaborative Embedded Systems

Collaborative embedded systems (CESs) are often safety-critical, operate in dynamic contexts within a collaborative system group (CSG), and must be capable of reacting to unforeseen situations without human intervention. Uncertainty that can occur during operation should be considered systematically during engineering to enable CESs to cope with uncertainties autonomously. This section first introduces an ontology for understanding uncertainty in CESs. It then continues with an explanation of different kinds of uncertainty using the platooning example.

7.1.1 Conceptual Ontology for Handling Uncertainty

Core concepts for describing uncertainty

In order to lay the foundation for understanding uncertainty that can occur during the operation of CESs, we have developed a lightweight ontology. This ontology defines core concepts for describing the different facets of uncertainty. It thereby provides the basis for seamless consideration of uncertainty in the engineering and operation of CESs. However, the ontological concepts are generic enough to describe different kinds of uncertainties, as we will see in Section 7.1.2. The ontology focuses on uncertainty that occurs during runtime, which differs from design-time uncertainty (e.g., ambiguous requirements) [Ramirez et al. 2012]. Nevertheless, the ontology is used during the design of CESs. Figure 7–1 shows the ontology as a UML class diagram.

Uncertainty related to information

As can be seen in Figure 7–1, *uncertainty* is the central concept of our ontology. However, other concepts are necessary to fully account for uncertainty and its consequences. Uncertainties are always related to certain *information* (e.g., about the operational context in which a CES is operating) that stems from a specific *source* (e.g., a sensor or another CES). With regard to information, uncertainty means that it is not clear whether the information is valid as processed in a specific *scenario* during the operation of CESs. In that scenario, *information* is processed by an *agent* to make a decision or to perform some activity based on the information. Hence, an agent perceives the uncertainty related to specific information. In order to cope with uncertainty, it needs to be quantified to allow for automated uncertainty handling approaches to be applied — for example, using probabilistic or fuzzy approaches.

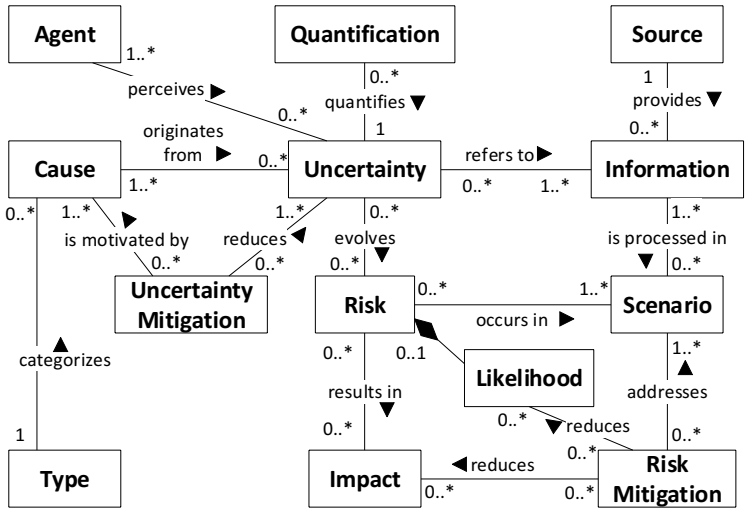


Fig. 7-1: *Core ontology for describing uncertainty*

Relationship between uncertainty and risk

In a *scenario*, *risks* may occur due to uncertainty. The term *risk* is used in a generic way here; it covers safety hazards but also, for example, economic risks. A risk can have a negative *impact*—for example, an accident—as well as a specific *likelihood* of occurrence. Uncertainty can be mitigated in two ways: 1) by mitigating the risk—that is, by reducing the likelihood or severity of an *impact*—or 2) by reducing the uncertainty that originally triggers the *risk*. Hence, the ontology includes the *risk mitigation* and the *uncertainty mitigation* concepts.

Cause and types of uncertainty

An uncertainty also always has at least one *cause* — that is, a reason why the uncertainty occurs. We can use the *type* concept to distinguish between different causes of uncertainty. We decided to relate the type of an uncertainty to its cause and not to the uncertainty itself, since categorizing uncertainties often relates to abstract sources of uncertainty, such as imprecision of sensors (cf., e.g., [Ramirez et al. 2012]), failures of communication networks, or insufficient trust in other agents (cf., e.g., [Yu et al. 2013]). Section 7.1.2 illustrates such types of uncertainty using the example of autonomous vehicles.

7.1.2 Different Kinds of Uncertainty

Uncertainty in "Cooperative Vehicle Automation"

In this section, we consider the use case of "Cooperative Vehicle Automation," which enables vehicles to form platoons on highways in order to reduce fuel consumption and increase traffic throughput (cf. [Jia et al. 2016]). As explained in Chapter 1, there are specific

www.dbooks.org

uncertainty-related challenges that emerge from this use case. In the following, we consider these challenges in more detail in order to illustrate different kinds of uncertainty that can occur during the operation of CESs. Specifically, we place particular emphasis on uncertainties rooted in the operational context of CESs, which is highly dynamic. The operational context comprises other CESs (e.g., other cars equipped with cooperative ACC (CACC) systems, which are engaged in or able to join a platoon) as well as non-collaborative context objects (e.g., pedestrians or vehicles steered manually).

System under consideration and context objects

An exemplary situation is depicted in Figure 7–2. It shows a platoon consisting of a certain number of vehicles (three of which are shown), as well as another non-collaborative vehicle driving ahead of the platoon leader (labeled "LV" in Figure 7–2). Furthermore, the car labeled "JV" attempts to join the platoon, which requires message exchange and coordination with the platoon leader. In the subsequent sections, we make use of this example to illustrate various concepts. Where applicable, we distinguish between the system under consideration (SUC) and its context objects (COs). For instance, in Figure 7–2, the platoon leader could be the SUC, whereas the messages it receives from other cars, as well as the other cars themselves and surrounding objects (e.g., road signs), are considered COs.

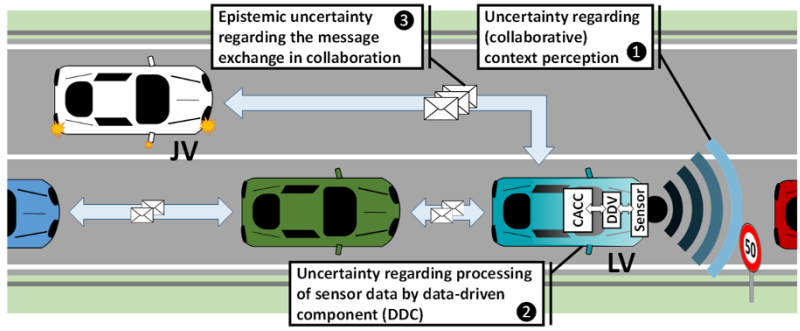


Fig. 7-2: *Uncertainty in the open and dynamic context of CESs*

Uncertainty relates to perception, data quality, and communication

There are three different sources of uncertainty under consideration:

1. *Perception-related uncertainty*: Due to the inherent imprecision of sensor devices, information representing the current context situation of a system may be invalid. CESs also share context information in order to jointly mitigate uncertainties. Each vehicle is equipped with sensory devices — such as a radar sensor or cameras for perceiving obstacles, other traffic participants, and

traffic signs. For example, an onboard camera sensor may be affected by dirt on the camera lens, resulting in uncertainty regarding the context information. Furthermore, the timing of events related to the context perception may cause uncertainties (e.g., caused by time delays in the transmission of sensor signals, see Section 7.2.2).

2. *Data-related uncertainty*: This kind of uncertainty is caused by limitations in the results of data-driven components (DDCs). These DDCs extract context knowledge from information sources (e.g., sensors). This context knowledge can then be processed further by some application logic (in this case, a CACC system). For instance, traffic signs detected by onboard cameras are recognized by embedded DDCs that contain artificial intelligence (AI) models that process camera images (see Section 7.3.2).

3. *Epistemic runtime uncertainty*: CESs exchange context information in order to reason about the current context situations they face. Uncertainty can be caused by epistemic concerns pertaining to the representation of the information that is exchanged and processed by CESs (see Section 7.3.1). For instance, context information about the goals of platoon members may be subject to ambiguity or incompleteness.

Complementary methods for handling uncertainty

Since these uncertainties have different causes (cf. Section 7.1.1), uncertainty is a multi-faceted and cross-cutting concern that requires a multitude of methods and techniques to enable CESs to handle such uncertainty autonomously during operation. The following sections present different methods for modeling and analyzing uncertainty that are complementary in our comprehensive uncertainty handling framework. Each method aims at handling a specific kind of uncertainty listed above. Furthermore, the different methods target both the development time (e.g., modeling methods) as well as runtime methods (e.g., wrapper components enabling the quantification of a DDC's uncertainty).

7.2 Modeling Uncertainty

In the following, we present approaches for modeling uncertainty, including a language for modeling early uncertainty information and relating it to other artifacts and a domain-specific method for capturing behavioral uncertainty in traffic scenarios.

www.dbooks.org

7.2.1 Orthogonal Uncertainty Modeling

In the engineering of CESs, uncertainties that may occur during operation must be considered systematically. In the early stages of requirements engineering in particular, potential runtime uncertainties must be identified and documented so that stakeholders can analyze and consider them appropriately. The ultimate goal of considering such uncertainty during engineering is to design CESs that are able to safely handle uncertainties during operation.

Orthogonality of uncertainty

Depending on project-specific demands, various different artifact types are used in a complementary manner to model a CES under development and other relevant information, such as the operational context. Typical modeling perspectives used in requirements engineering distinguish structural, functional, and behavioral aspects of the CES and its context. Uncertainties are not limited to a single artifact but rather affect different aspects modeled in several artifacts. For example, perception-related uncertainty due to the failure of a sensor (see Section 7.1.2) is reflected in all three modeling perspectives: the interface between the CES and the sensor device—where the uncertainty appears in the form of missing or corrupted data—is captured in structural models; sensor data is processed by CES functions modeled in the functional perspective; and finally, sensor failure may result in certain (mis-)behavior, where uncertainty can be detected, and uncertainty may affect the CES behavior.

Difficulties in capturing uncertainty information

Hence, uncertainty can be considered a cross-cutting concern, similar to variability (cf., e.g., [Bachmann et al. 2003], Chapter 18). Describing uncertainty by attaching uncertainty information to model elements across different artifacts (e.g., denoting sensor data as the information subject to uncertainty) has two essential drawbacks:

1. The information necessary to specify uncertainty is documented redundantly at multiple locations attached to model elements. For instance, sensor data may be modeled in structural models and as input for functions in functional models.

2. Uncertainty information is spread across several models. This makes it difficult to structure uncertainty information and trace essential parts of an uncertainty description (e.g., the cause of uncertainty) throughout the various artifacts. Furthermore, it is difficult to capture relationships between different uncertainties.

Orthogonal uncertainty models

In the following, we present a modeling technique [Bandyszak et al. 2018b], [Bandyszak et al. 2020] to support the systematic identification and documentation of runtime uncertainties during the

entire engineering process. The core idea is to capture information that describes uncertainty (see Section 7.1.1) in a separate artifact and connect this information to other engineering artifacts (called "base artifacts") through trace links. These first-class uncertainty artifacts created using a dedicated graphical modeling language are called "*orthogonal uncertainty models*" (OUMs). OUMs provide a central artifact for analyzing uncertainties and how they affect and must be accounted for during engineering.

Modeling Concepts and Notation

Graphical modeling language

OUMs employ a specific perspective on the operation of an SUC. The modeling concepts and the corresponding graphical notation are introduced in Table 7-3. As we can see, the core concepts for describing uncertainty are captured using UML-like stereotype notation and specific iconic representations to foster understanding of the uncertainty information modeled. The visual notation comprises node types and connector elements for modeling uncertainty relationships within the OUM, as well as establishing relationships to other artifacts.

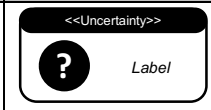
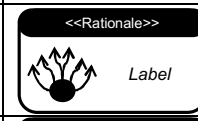
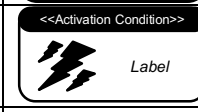
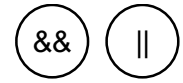
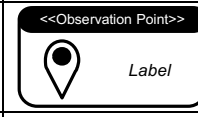
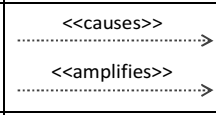
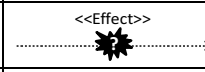
Modeling uncertainty observation and cause

In order to model uncertainty from the dedicated perspective of a CES during operation (i.e., the "agent" under consideration), we define some specialized extensions to the ontology defined in Section 7.1.1. While the actual information subject to uncertainty is captured in the base artifacts, the OUM uses the *observation point* concept to specify where uncertainty becomes visible. Furthermore, the cause of an uncertainty is subdivided into its *rationale* (i.e., the reason why uncertainty occurs) and *activation condition* elements. Rationales and activation conditions can be combined in a tree-like structure using conjunction/disjunction nodes to describe the cause of uncertainty.

Modeling uncertainty relationships

The possible effects of an uncertainty (i.e., risks and impacts of uncertainty, see Section 7.1.1) are captured using *effect* links that point to base artifact elements — for example, system functions affected by uncertainty. In contrast, *uncertainty mitigations* are modeled in the base artifacts (e.g., specific functions that cope with uncertainty) and related to the uncertainty modeled in OUMs. The OUM also allows relationships between different uncertainties to be modeled, including causal and amplification relationships (see Table 7-3).

www.dbooks.org

Modeling element	Explanation	Visual notation
Uncertainty	This element is used to capture the uncertainty itself and identify it, which allows related uncertainty information (further specified by the other model elements) to be grouped and also allows distinction between different uncertainties.	
Uncertainty relation	This connection node visualizes trace relationships of the concepts outlined above by means of an n-ary association.	
Uncertainty element connector	This edge type establishes relationships between *uncertainty* and *observation point* elements as well as uncertainty causes (causal combinations of *rationale* and *activation condition* elements).	
Rationale	This element describes the root cause of the uncertainty — i.e., the source that originally introduces uncertainty.	
Activation condition	This element describes the condition under which an uncertainty can become active during operation.	
Uncertainty cause conjunction/ disjunction	The cause of an uncertainty can be considered a causal (AND/OR) combination of a set of *uncertainty rationales* and a set of *activation conditions*. These manifest in a binary tree with two kinds of control nodes.	
Observation point	This element documents the artifacts in which an uncertainty is present and where the system can detect the said uncertainty.	
Uncertainty relations	These relationships document situations where one uncertainty can cause or amplify another uncertainty (i.e., when active, it may sustain effects and thereby complicate the mitigation of the amplified uncertainty).	
Trace relation	This relationship is used to trace uncertainty information to base artifact elements.	
Effect relation	This relationship refers to the effect an uncertainty has on either the SUC or its context.	
Mitigation relation	This relationship, pointing from a base artifact element to a corresponding OUM uncertainty element, captures possible mitigations that either counteract uncertainty once it has occurred (i.e., to reduce possible negative effects), or prevent uncertainty from becoming active.	

Tab. 7-3: *OUM concepts and notation [Bandyszak et al. 2020]*

Example

Figure 7–3 shows an example OUM (middle part) as well as some exemplary base artifacts (upper and lower part). The latter include a structural context model, a functional model, and a behavioral model of the exemplary CACC system. To reduce complexity, only excerpts related to the joining maneuver are depicted. Moreover, a sequence

diagram shows interactions between two vehicles—each equipped with a CACC—in the course of the joining maneuver.

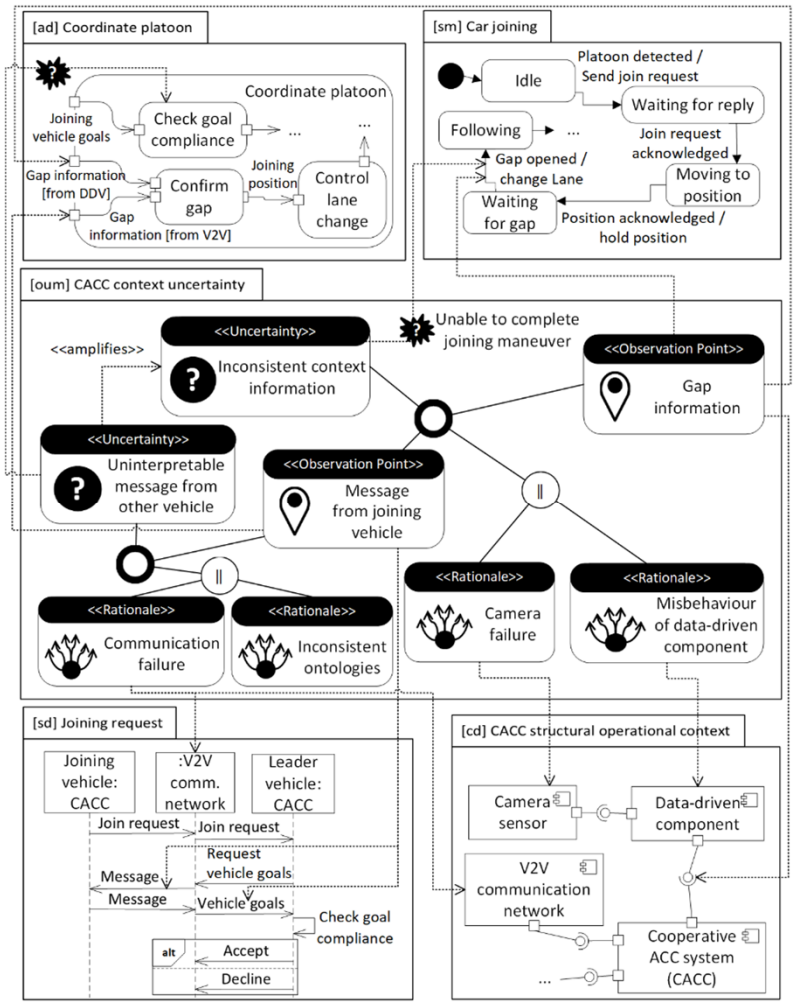


Fig. 7–4: *Orthogonal uncertainty model example*

Inconsistent information from different sources

The exemplary OUM specifies two uncertainties. First, there can be uncertainty related to inconsistent context information. In this example, the context is perceived in the form of camera data processed by a DDC (cf. Section 7.3.2), as well as by context information communicated by other vehicles, because in the joining maneuver, the leader vehicle informs the joining vehicle about the gap to be taken. The uncertainty of inconsistent information occurs when the gap information obtained by the DDC (e.g., indicating an obstacle

www.dbooks.org

in the gap) differs from the information communicated by the platoon leader (e.g., gap is free). The rationale of this uncertainty could be a failure of the camera sensor or misbehavior of the DDC that interprets the camera data. A potential effect may be the inability to safely complete the joining maneuver. This uncertainty is described further in [Bandyszak et al. 2018a].

Uncertainty in inter-vehicle communication

Another uncertainty modeled in this exemplary OUM relates to uninterpretable messages from other vehicles. In Figure 7–4, this message exchange is illustrated in a sequence diagram. This uncertainty may be caused by technical communication failures of the V2V network, or inconsistent ontologies underlying the representation of the data that is exchanged (cf. Section 7.3.1). For example, during the join maneuver, goals (e.g., target destination and driving preferences) must be exchanged between the vehicle that requested to join a platoon and the platoon leader. However, when inconsistent ontologies are used to specify goals (e.g., considering different concepts for electric vehicles), the goals of the joining vehicle cannot be interpreted, which in turn affects the corresponding CACC function.

7.2.2 Modeling Uncertainty in Traffic Scenarios

When designing a CES, it is important to estimate the impact of uncertainties (as early as possible) in order to be able to make qualified design decisions. Therefore, semi-formal uncertainty modeling techniques (such as the OUM presented in Section 7.2.1), which are especially helpful for understanding uncertainty in early design phases, must be complemented with formal modeling approaches, which allow quantification of the possible influence on CSG behavior throughout the design process.

Formal modeling of uncertainties

For this aim, uncertainties must be modeled mathematically—for example, as random variable or probabilistic distribution—and the effects of uncertainties on behavior must be identified and modeled, resulting in probabilistic CES/CSG behavior models. Probabilistic behavior models can be used as a base for mathematical analysis — for example, by probabilistic model checking or stochastic analysis (e.g., by using repeated simulation).

Assessing uncertainty impact according to defined criteria

Within the mathematical analysis, the probabilistic behavior model is assessed with regard to an evaluation criterion within the context of a scenario. Typical evaluation criteria are risk or quality of service. Mathematical assessment of the impact of uncertainties can help to guide the design process in many areas, including:

1. Estimating the kind of mitigation required for uncertainties (or the degree of weakening the influence of uncertainties, if they cannot be completely mitigated)
2. Developing and choosing more uncertainty resilient implementations and strategies
3. Assessing the fulfilment of safety requirements (especially if homologation is required)

Scenario-based behavioral modeling

Summarizing the above, for behavioral modeling of uncertainties, a behavior model is extended to a probabilistic or stochastic behavior model. For assessment, a scenario model and an evaluation criterion are also required. These model components are described in more detail in the following. We start with scenario modeling, which is necessary to describe the situation for which the assessment is performed.

Modeling Traffic Scenarios for CSGs

Modeling a traffic scenario with the participation of CSGs requires a specification language with the ability to express dynamically changing relationships (including spatial relationships) between a variable number of CESs involved in a CSG.

Traffic sequence charts (TSCs)

Traffic sequence charts (TSCs) [Damm et al. 2017], [Damm et al. 2018] are one such formalism that allow intuitive and concise graphical specification of dynamic constraints based on formal semantics. TSCs (which—with a different focus—are also described in Chapter 6) are based on acyclic graphs of chart nodes. Chart nodes capture constraints over a time interval and can be combined into a chart using sequence, choice, or parallel composition. In a TSC, the constrained time intervals are seamless — that is, there is no time gap within a sequence of two nodes.

Chart nodes include simple invariants (constraints that hold throughout the complete constrained interval) and conditions (constraints holding at least once), as well as complex nodes that specify communication/event patterns or contain complete charts. Spatial views describe spatial constraints between objects, represented by symbols in a topological view. Spatial views can be used as base constraints for invariant and condition nodes.

TSC example

For example, the TSC excerpt in Figure 7-5 describes a scenario in which the car with index n+1 joins a platoon consisting of the vehicles with index 1 (platoon head) to n (platoon tail) driving at speed v. The TSC consists of four chart nodes: the first chart node is an invariant

www.dbooks.org

that contains a spatial view and describes the phase where the approaching car is still far away from the platoon (which is represented by an abstract symbol, see Chapter 6 for more detail). Phase 2 is described by two nodes holding in parallel: the spatial invariant expresses that the approaching car is closer to the platoon but has not yet reached the distance for initiating a join maneuver. During phase 2, the approaching car sends a join request to the last car of the platoon, which eventually answers granting the right to join. In phase 3, the approaching car actually joins the platoon, and then phase 4 is reached, in which the car is part of the platoon.

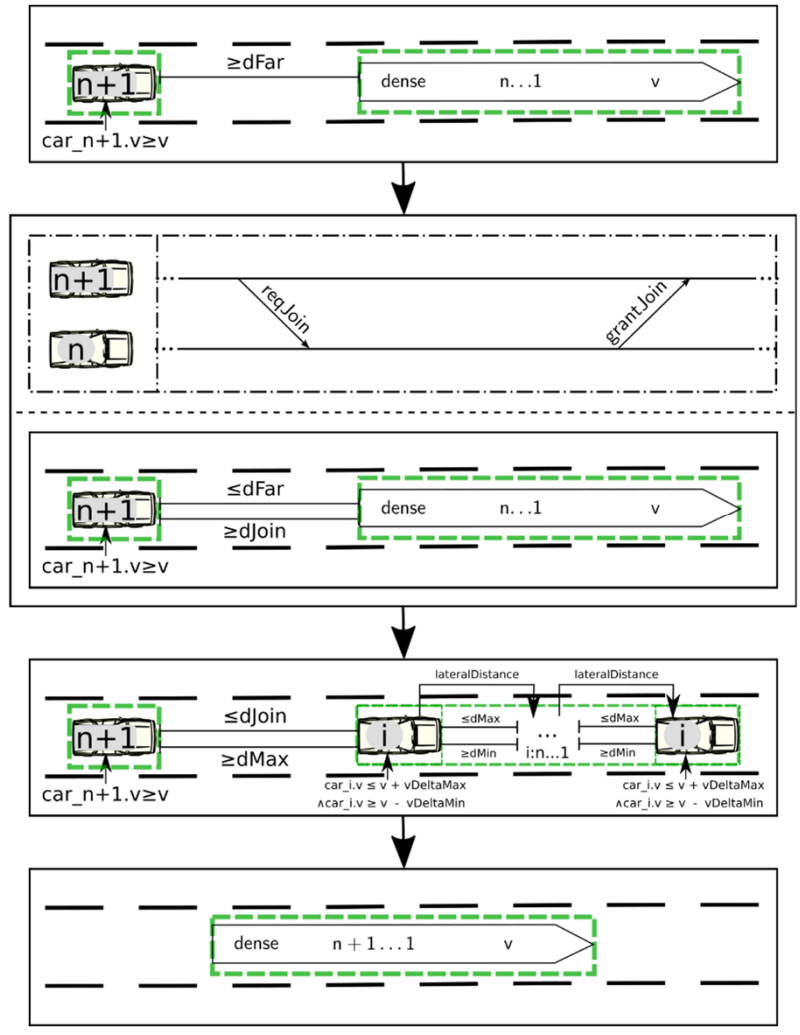


Fig. 7–5: *TSC excerpt describing a car joining a platoon*

World models

TSCs are interpreted with respect to a modular world model, which defines object classes, interfaces, and behaviors. World model modules are exchangeable, as long as suitable interfaces are provided. Consequently, behavior modules can be specified using different specification languages (with suitable interfaces) — for example, hybrid automata or stochastic/probabilistic hybrid automata. This flexibility can be used for combining the same scenario specification with different CES implementation variants or CESs whose granularity has been refined during the development process.

Behavioral Uncertainty Modeling

Considering uncertainty related to timing

As a first step for behavioral uncertainty description, the occurrences of uncertainties must be identified and modeled mathematically. For behavioral modeling, it is mostly runtime uncertainties—which influence the behavior of the model—that are relevant. These uncertainties can be separated into two classes, depending on the timing of interactions. *Discrete time uncertainties* deal with discrete events, such as sending and receiving messages, or sensors resulting in discrete signals (see Section 7.1.2). *Continuous time uncertainties* deal with continuous interactions, such as following a trajectory or sensors continuously reporting the location of objects (at least the start and end of object detection are discrete events). Both classes include uncertainties due to time delays and data distortion, but only discrete time events can be completely missed.

Discrete time and continuous time uncertainties

Discrete time event misses can be modeled mathematically by a random variable connected with a probability. Data distortions occur with a certain probability distribution. A discrete time delay can be modeled by a random variable with a probability distribution. Time delays for continuous uncertainties may involve time compaction or dilation and are a little more complicated, but can still be covered by a (more complex) probability distribution.

These mathematical uncertainty models can be embedded in hybrid automata behavior models, resulting in probabilistic hybrid automata incorporating uncertainty behavior.

Risk Assessment

Probabilistic and stochastic model checking

The last missing building block describes the use of the probabilistic behavioral CES/CSG model involving uncertainty. There are two main analysis methods: *probabilistic model checking* [Katoen 2016] can compute the probability with which an evaluation criterion (such as a criticality or a quality of service measure) is reached. Probabilistic

www.dbooks.org

model checking is mathematically more stable but is more applicable for small systems due to its complexity. *Stochastic model checking* [Kwiatkowska et al. 2007] (which is done by performing many simulation runs) is not as exact, but the performance is less challenging. During simulation, the probabilistic elements of the probabilistic behavior model must be instantiated according to the associated probability distribution. The resulting simulation runs are then assessed with respect to the evaluation criterion. Similar to probabilistic model checking, the results can be used as a quality measure for the behavior modeled.

7.3 Analyzing Uncertainty

In addition to modeling uncertainty semi-formally and formally, more specific analysis methods are necessary to fully account for the various kinds of uncertainty (see Section 7.1.2). Uncertainty analysis methods cover, among other things, guidance for identifying potential uncertainty sources as well as eliciting more specific information for (automatically) estimating uncertainty. In the following, we present a classification scheme that helps identify epistemic uncertainties, as well as an approach to provide situation-aware estimates of the uncertainty in an outcome of a DDC during operation of CESs.

7.3.1 Identifying Epistemic Uncertainties

This section presents a classification scheme that facilitates the identification and classification of epistemic uncertainties resulting from information exchange in CSGs (see Section 7.1.2). For this purpose, we first describe the different types of epistemic uncertainties that may occur in this context. We then present the epistemic uncertainty classification scheme for runtime information exchange (EURECA) and its application during requirements engineering. For an in-depth presentation of the approach, including formal modeling of knowledge and epistemic uncertainties (as well mitigation strategies, etc.), please refer to [Hildebrandt et al. 2019].

Epistemic uncertainties in platooning

In this section, we use an example in which the SUC is a platoon leading vehicle that runs on a combustion engine and the CO is an electric vehicle that wants to join the platoon (see Section 7.1.2). As epistemic uncertainties result only from a lack of understanding regarding a message the SUC *receives*, all examples concern messages sent from the CO to the SUC. As we show in the following, epistemic

uncertainties can appear at the type level and at the instance level. Uncertainty sources at the type level are rooted in the terminological knowledge utilized by CESs to specify messages. In simple terms, terminological knowledge is comparable to a vocabulary that includes the relationships between vocabulary items and is from here on referred to as the TBox [Krotzsch et al. 2014]. At each level—that is, type and instance level—four different classes of uncertainties can be distinguished. Figure 7–6 illustrates a subset of these classes (T1, T3, I1, and I2). As we can see, the set difference of the SUC TBox (upper ellipse at the top of Figure 7–6) and the CO TBox (lower ellipse) is not empty, resulting in various uncertainties. The uncertainty classes illustrated and additional classes are detailed in the following.

Uncertainty Sources at the Type Level

There are four classes of uncertainties that may occur at the type level. All of these uncertainties result from a mismatch between the TBox of the SUC and the CO used for specifying and interpreting the messages to be exchanged at runtime.

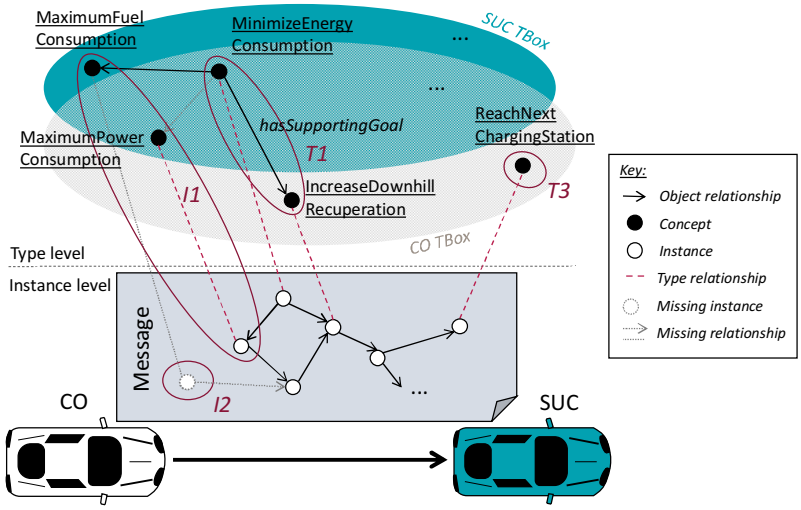


Fig. 7–6: *Epistemic uncertainties at type and instance level*

Known difference in scope

The first class of type-level uncertainties results from a *known difference in scope* (T1) between the TBox of the SUC and the CO. This uncertainty occurs whenever the CO sends the SUC a message that includes a TBox element that is not included in the SUC's TBox. The difference in scope is *known* insofar as the *unknown* TBox element has a *known* relationship to a *known* TBox element, which may allow for

www.dbooks.org

some kind of mitigation at runtime. For instance, if the message contains an element "MinimizeEnergyConsumption" with the relationship "hasSupportingGoal" connecting it to the element "IncreaseDownhillRecuperation," the SUC may understand the first two elements but not the last one since it is not an electric vehicle.

Unknown difference in scope

The second class occurs whenever there is an *unknown difference in scope* (T2). In this case, the message contains an *unknown* TBox element that has an *unknown* relationship to a *known* TBox element. This makes mitigation at runtime more challenging. For instance, if the message from the CO contains an element "DesiredStateAtDestination" with the relationship "hasStateOfCharge" to the element "StateOfCharge," the SUC may understand the first element but not the second and third as these are concepts specific to an electric vehicle.

Distinct ontologies scope

The third class of type-level uncertainties results from a *distinct scope* (T3). In this case, the *unknown* TBox element has no attachment to the SUC's TBox whatsoever and mitigation is barely possible. For instance, a message from the CO containing the goal "ReachNextChargingStation" might not be understood by the SUC at all because it has no relationship to any known concept.

Inconsistent ontological commitment

Uncertainties of the fourth class occur whenever a message indicates an *inconsistent ontological commitment* (T4). For instance, if the CO informs the SUC about its consumption in the measurement unit kWh, whereas the SUC measures its consumption in liters, this may lead to uncertainty even if the SUC's TBox contains the elements kWh and consumption.

Uncertainty Sources at the Instance Level

While the previously described type-level uncertainties result from terminological differences, the following uncertainties pertain to the actual information contained in a message. The first class, *semantically inconsistent information* (I1), occurs when a message contains a value that violates the semantic definition of a relationship that it refers to. For example, the SUC is informed by the CO about its "MaximumPowerConsumption" with regard to the goal "MinimizeEnergyConsumption." Even if the SUC understands the former concept in a different context (such as its power outlet), this message will result in uncertainty as the SUC running a combustion engine would consider only "MaximumFuelConsumption" a valid concept in this context.

Situationally incomplete information

Situationally incomplete information (I2) as a source of uncertainty occurs whenever a message contains a set of statements that does not satisfy the requirements of the situation at hand. For instance,

consider a CO that wants to join a platoon and the SUC as the platoon leader requires the information "MaximumFuelConsumption." However, if the CO provides only the "Destination" in its message, the SUC will perceive the message as incomplete.

Situationally inconsistent information

The third class of instance-level uncertainties results from *situationally inconsistent information* (I3), which occurs whenever the content of a message is inconsistent with regard to the information expected by the SUC for the situation at hand. Consider a platoon where the vehicles regularly broadcast their range in kilometers. If the platoon now descends a steep road, the virtually constant braking might actually increase the reported range of the CO (the electric vehicle). The SUC, however, might expect a range decrease over time and thus considers the reported range to be *situationally inconsistent*.

Missing type membership

The fourth class of instance-level uncertainties results from *missing type membership* (I4). This class of uncertainty occurs when a message contains an information item that lacks a type membership. For instance, the SUC might receive geographical coordinates from a CO but no information on what these coordinates refer to.

EURECA

In order to systematically analyze and capture the previously described uncertainties at the instance and type level, we developed an epistemic uncertainty classification scheme for runtime information exchange (EURECA). The two-dimensional scheme is depicted in Table 7-7.

SUC ontology	Ontology element	Type-level uncertainty				Instance-level uncertainty			
		T1	T2	T2	T4	I1	I2	I3	I4
Goal ontology	hasSupportingGoal	x							
	DesiredStateAtDestination		x						
	ReachNextChargingStation			x					
	hasConsumption				x				
	MaximumPowerConsumption					x			
	MaximumFuelConsumption						x		
	Range							x	
	GeographicalCoordinates								x

Tab. 7-7: *EURECA applied to the platooning example*

Two-dimensional classification scheme

The first column is populated with the SUC ontology that the uncertainties captured relate to. In our examples, we considered only the SUC's goal ontology, but other ontologies obviously might be subject to epistemic uncertainties as well. The second column lists the concrete ontology elements that are subject to epistemic uncertainty according to the analysis performed. A checkmark indicates which

www.dbooks.org

specific epistemic uncertainty has been identified for an ontology element. Note that in our example, each ontology element is subject to exactly one type of epistemic uncertainty; however, the elements could also be subject to multiple types. Instead of checkmarks, we could also indicate the specified communication scenario in which an element is subject to uncertainty (cf. [Hildebrandt et al. 2019]). This approach is generally recommended due to improved traceability when new scenarios are added or scenarios are changed during subsequent requirements engineering iterations — for example, when an agile development approach is applied.

The epistemic uncertainty classes identified are rooted in an analysis of the underlying ontological foundations of knowledge exchange and are not the result of domain-specific considerations. Hence, while evaluation has been performed only for the automotive domain, we are confident that EURECA can also be applied to other domains — for example, distributed energy production and adaptive factories.

7.3.2 Assessing Data-Driven Uncertainties

Components with data-driven models [Solomatine and Ostfeld 2008], such as those obtained when applying AI and machine learning methods, are becoming increasingly important for complex software-intensive systems. In particular, CESs intended to collaborate in an open context have to process various kinds of sensor inputs to recognize and interpret their situation in order to handle changes in their environment and interact with previously unknown agents. Unlike traditionally engineered components, which are developed by software engineers who define their functional behavior using code or models, the functional behavior of a DDC is learned automatically from gathered data through an algorithm.

Assuring correct behavior of a DDC

As a consequence, the functional behavior expected from a DDC can be specified only in part on its intended application domain, usually based on a number of example cases for which data was previously collected and augmented with additional information. Consequently, we cannot provide assurance that such a component will behave as intended in all cases [Kläs 2018], resulting in data-related uncertainty (see Section 7.1.2).

Example situation

In our example, the current speed limits have to be considered when a vehicle wants to join an existing platoon (cf. Figure 7–2). One *information source* for current speed limits can be a DDC that provides traffic sign recognition (TSR) based on data of the camera sensor in

the CESs of interest (as shown in Figure 7–3). At the latest since 2012, deep convolutional neural networks have proven their superior performance for this kind of task and can be considered as a state-of-the-art approach for TSR [Krizhevsky et al. 2017]. Nevertheless, uncertainty remains inherent in the outcome of our TSR component since we cannot specify for all possible combinations of pixel values within an image what kind of traffic sign should be reported.

Considering both design and runtime of DDCs

Because the use of DDCs is an important source of uncertainty in CESs, the uncertainty they introduce must be appropriately understood and managed not only during *design time* but also during *operation*. In the following, we first present a classification for the different sources of uncertainty relevant when applying a DDC, and then introduce "*uncertainty wrappers*" as a means of quantifying and analyzing the level of uncertainty for any specific situation at runtime.

Three Types of Uncertainty Sources

The sources of uncertainty in DDCs can be separated into three major *types*: uncertainties caused by limitations in terms of model fit, data quality, and scope compliance [Kläs and Vollmer 2018]. Whereas *model fit* focuses on the inherent uncertainty in data-driven models, *data quality* covers the additional uncertainty caused by their application to input data obtained in suboptimal conditions, and *scope compliance* covers situations where the model is probably applied outside the scope for which the model was trained and validated.

Examples for uncertainty sources

In our example, limitations in model fit may be caused by restrictions in the number of model parameters, input variables considered, and data points available to train the model. Limitations in data quality may be caused by weather conditions, such as rain, natural and artificial backlight, a dirty camera lens, and other factors that make it more difficult for the TSR component to reliably recognize the correct traffic sign on the given image. Finally, limitations in scope compliance occur when a model is used outside its target application scope (TAS). For example, if our TSR component was intended to be used for passenger cars in Germany, its application in a different country will make its outcomes questionable because the component was most likely built and validated only for German traffic signs.

Managing Uncertainty during Operation

The *uncertainty wrapper* approach was developed to better deal with uncertainty inherent in the outcomes of data-driven models during

www.dbooks.org

operation. It supports this purpose by encapsulating the data-driven model to enrich the model outcomes with dependable and situation-aware estimates of uncertainty. This allows the CES that processes these outcomes to make more informed and dependable decisions.

Characteristics of uncertainty wrappers

The approach is *holistic* in the sense that it addresses uncertainty caused by limitations in model fit, data quality, and scope compliance. Moreover, it is *generic* in the sense that it is model agnostic — that is, it states no requirements on the data-driven model it encapsulates. Specifically, it does not require that the existing model be adapted nor that the model provides specific kinds of outcomes.

The uncertainty wrapper approach consists of a selection of concepts that extend and refine the uncertainty ontology introduced in Section 8.1, equations that allow quantification and aggregation of uncertainties from different types of sources [Kläs and Sembach 2019], an architectural design proposal, and a tooling framework for building and validating uncertainty wrappers [Kläs and Jöckel 2020].

Uncertainty Wrapper – Architecture and Application

In the following, we introduce the most prominent elements of the uncertainty wrapper architecture and relate them. Moreover, we illustrate them on a simplified application example in the context of the previously introduced example case.

Extension of DDC outcomes

The wrapper encapsulates the existing data-driven model and extends its *outcome* with *dependable uncertainty estimates* (Figure 7–8).

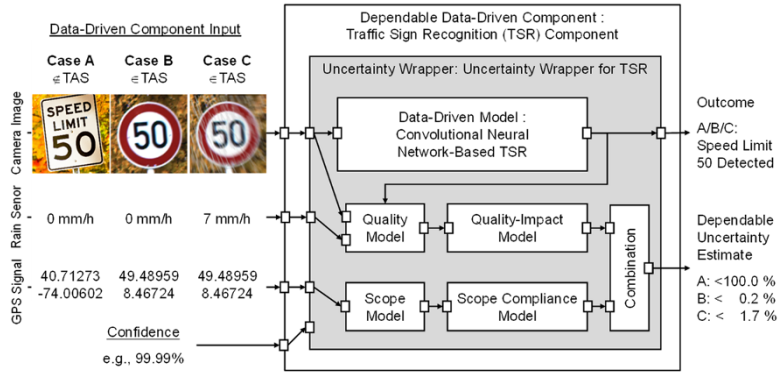


Fig. 7-8: *Uncertainty wrapper architecture and example applications*

Example for application

In our example, the outcome of the model could be the information about whether the data-driven model has recognized a “speed limit

50" sign or not (cf. Figure 7–2). The uncertainty is then expressed by the likelihood that the outcome provided is not correct.

Further data to assess scope compliance

Besides the data processed by the data-driven model, the *data-driven model input* may contain further data that is used by the wrapper to assess the degree of uncertainty in the model outcome. For example, the GPS signal may be used to determine whether the vehicle is still in Germany, a task which is conducted by the *scope model*. The result is then provided as a scope factor evaluation result to the *scope compliance model,* which calculates the likelihood of scope incompliance considering the results for all scope factors.

Quality and quality impact model

Moreover, the rain sensor signal may be also used as an input. Based on this signal, the *quality model* determines the level of precipitation, which is anticipated to have an influence on the recognition performance of the data-driven model. Together with the results of other quality factors, the *quality impact model* then determines a situation-aware uncertainty estimate using a decision tree model as a white box approach.

Finally, the uncertainty information provided by the scope compliance and quality impact model are combined to give a dependable uncertainty estimate that considers the requested level of *confidence*.

Examples for application results

In Figure 7–8, we illustrate this for three cases. In Case A, we get an extremely high uncertainty because the DDC is used in New York, which is outside the TAS. In Case B, we obtain low uncertainty since the component is used in its TAS under good quality conditions. In Case C, we would end up with a moderate uncertainty since the rain makes the recognition task more difficult.

Uncertainty Wrappers – Limitations and Advantages

In order to build an uncertainty wrapper, in addition to the existing data-driven model, we need data for a representative set of cases from the TAS, most preferably augmented with labels that indicate quality deficits. Alternatively, quality deficit augmentation frameworks such as the one presented in [Jöckel and Kläs 2019] may be applied.

Unbiased and broader uncertainty assessment

In comparison to a more traditional approach, where uncertainty is estimated by the data-driven model itself, the use of uncertainty wrappers provides advantages in practice. Its uncertainty estimates are unbiased since it does not rely on data used during the training of the data-driven model. Moreover, the uncertainty assessment is broader because it considers more types of uncertainty sources, including the commonly neglected scope compliance, and also information that is relevant for uncertainty but not for the primary

www.dbooks.org

objective of the data-driven model. Finally, uncertainty wrappers can simplify safety assessments of data-driven models by not only providing statistically safeguarded uncertainty estimates at a requested confidence level, but also promising more comprehensible evaluations by domain and safety experts since they decouple the uncertainty analysis from the "black box" as which most data-driven models are still considered.

7.4 Conclusion

CESs operate in highly dynamic contexts and thus have to cope with uncertainties during operation. This uncertainty cannot be fully anticipated during design, since CESs are increasingly autonomous and adaptive, and CSGs may take various forms. Nevertheless, it is crucial to systematically consider potential runtime uncertainties during the engineering of CESs. This requires methods for identifying, modeling, and analyzing uncertainty to develop CESs capable of coping with uncertainty during operation autonomously.

Uncertainty ontology

This chapter approached the complex task of uncertainty handling during CES engineering by first conceptualizing uncertainty. To this end, we presented an uncertainty ontology that defines core concepts to describe uncertainty. Among other aspects, this ontology provides a means of describing and understanding causes of uncertainty and relating uncertainty to information gathered through certain sources (e.g., sensors) and processed by CESs.

Versatile modeling and analysis methods

Based on the ontology and a characterization of different kinds of uncertainty relevant for CESs and CSGs, we presented methods for modeling uncertainty, and for analyzing uncertainty and its effects during both design and operation. To model uncertainty graphically, especially during early phases, we presented a modeling language for specifying uncertainty in dedicated artifacts — that is, orthogonal uncertainty models. As a more formal approach, we presented a behavioral modeling approach based on traffic sequence charts for analyzing risks in dynamic traffic scenarios. As part of the analysis methods, we presented a classification scheme for identifying epistemic uncertainties in the information exchange among CESs. Furthermore, we presented an analysis method to support the safe operation of DDCs by equipping them with a wrapper that provides reliable information on situation-specific uncertainty.

7.5 Literature

[Bachmann et al. 2003] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, A. Vilbig: A Meta-model for Representing Variability in Product Family Development. In: Software Product-Family Engineering. LNCS Vol. 3014, Springer, 2003, pp. 66–80.

[Bandyszak et al. 2018a] T. Bandyszak, P. Kuhs, J. Kleinblotekamp, M. Daun: On the Use of Orthogonal Context Uncertainty Models in the Engineering of Collaborative Embedded Systems. In: Joint Proceedings of the Workshops at Modellierung 2018. CEUR Vol. 2060, 2018. pp. 121–130.

[Bandyszak et al. 2018b] T. Bandyszak, M. Daun, B. Tenbergen, T. Weyer: Model-Based Documentation of Context Uncertainty for Cyber-Physical Systems. In: Proc. IEEE 14th Int. Conference on Automation Science and Engineering (CASE). IEEE, 2018, pp. 1087–1092.

[Bandyszak et al. 2020] T. Bandyszak, M. Daun, B. Tenbergen, P. Kuhs, S. Wolf, T. Weyer: Orthogonal Uncertainty Modeling in the Engineering of Cyber-Physical Systems. In: IEEE Transactions on Automation Science and Engineering, Vol. 17, No. 3, 2020, pp. 1250–1265.

[Damm et al. 2017] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, A. Rakow: Traffic Sequence Charts – From Visualization to Semantics. Technical Report 14 AVACS, No. 117, 2017.

[Damm et al. 2018] W. Damm, E. Möhlmann, T. Peikenkamp, A. Rakow: A Formal Semantics for Traffic Sequence Charts. In: Principles of Modeling – Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday, LNCS, Vol. 10760, 2018, pp. 182–205.

[Hildebrandt et al. 2019] C. Hildebrandt, T. Bandyszak, A. Petrovska, N. Laxman, E. Cioroaica, S. Törsleff: EURECA: Epistemic Uncertainty Classification Scheme for Runtime Information Exchange in Collaborative System Groups. SICS Software-Intensive Cyber-Physical Systems, Vol. 34, No. 4, 2019, pp. 177–190.

[Jia et al. 2016] D. Jia, K. Lu, J. Wang, X. Zhang, X. Shen: A Survey on Platoon-Based Vehicular Cyber-Physical Systems. IEEE Communications Surveys Tutorials, Vol. 18, No. 1, 2016, pp. 263–284.

[Jöckel and Kläs 2019] L. Jöckel, M. Kläs: Increasing Trust in Data-Driven Model Validation – A Framework for Probabilistic Augmentation of Images and Meta-Data Generation Using Application Scope Characteristics. In: 38th Int. Conf. Computer Safety, Reliability and Security (SafeComp). Springer, 2019, pp. 155–164.

[Katoen 2016] J.-P. Katoen: The Probabilistic Model Checking Landscape. In: Proc. 31st Ann. ACM/IEEE Symp. Logic in Computer Science (LICS). ACM, 2016, pp. 31–45.

[Kläs 2018] M. Kläs: Towards Identifying and Managing Sources of Uncertainty in AI and Machine Learning Models - An Overview. arXiv preprint, arXiv:1811.11669.

[Kläs and Jöckel 2020] M. Kläs, L. Jöckel: A Framework for Building Uncertainty Wrappers for AI/ML-Based Data-Driven Components. In: 3rd Int. Workshop on Artificial Intelligence Safety Engineering (WAISE) - submitted, 2020.

[Kläs and Sembach 2019] M. Kläs, L. Sembach: Uncertainty Wrappers for Data-Driven Models – Increase the Transparency of AI/ML-Based Models through Enrichment with Dependable Situation-Aware Uncertainty Estimates. In: 2nd Int. Workshop Artificial Intelligence Safety Engineering (WAISE). Springer, 2019, pp. 358–364.

www.dbooks.org

[Kläs and Vollmer 2018] M. Kläs, A. M. Vollmer: Uncertainty in Machine Learning Applications – A Practice-Driven Classification of Uncertainty. In: 1st Int. Workshop Artificial Intelligence Safety Engineering (WAISE). Springer, 2018, pp. 431–438.

[Krizhevsky et al. 2017] A. Krizhevsky, I. Sutskever, G. E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, Vol. 60 No. 6, 2017, pp. 84–90.

[Krotzsch et al. 2014] M. Krotzsch, F. Simancik, I. Horrocks: Description logics. IEEE Intelligent Systems, Vol. 29, No. 1, 2014, pp. 12–19.

[Kwiatkowska et al. 2007] M. Kwiatkowska, G. Norman, D. Parker: Stochastic Model Checking. In: Formal Methods for Performance Evaluation. Springer, 2007, pp. 220–270.

[Ramirez et al. 2012] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng: A Taxonomy of Uncertainty for Dynamically Adaptive Systems. In: 7th Int. Symp. Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE, 2012, pp. 99–108.

[Solomatine and Ostfeld 2008] D. Solomatine, A. Ostfeld: Data-Driven Modelling: Some Past Experiences and New Approaches. Journal of Hydroinformatics, Vol. 10, No. 2, 2008, pp. 3–22.

[Yu et al. 2013] H. Yu, Z. Shen, C. Leung, C. Miao, V. R. Lesser: A Survey of Multi-Agent Trust Management Systems. IEEE Access, Vol. 1, 2013, pp. 35–50.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



David Santiago Velasco Moncada, Fraunhofer IESE
Daniel Schneider, Fraunhofer IESE
Ana Petrovska, Technical University of Munich
Nishanth Laxman, Technical University of Kaiserslautern
Felix Möhrle, Technical University of Kaiserslautern
Stefan Rothbauer, Siemens AG
Marc Zeller, Siemens AG
Chee Hung Koo, Robert Bosch GmbH
Samira Safdari, Expleo Germany

8

Dynamic Safety Certification for Collaborative Embedded Systems at Runtime

Traditionally, integration and quality assurance of embedded systems are done entirely at development time. Moreover, since such systems often perform safety-critical tasks and work in human environments, safety analyses are performed and safety argumentations devised to convince certification authorities of their safety and to certify the systems if necessary. Collaborative embedded systems, however, are designed to integrate and collaborate with other systems dynamically at runtime. A complete prediction and analysis of all relevant properties during the design phase is usually not possible, as many influencing factors are not yet known. This makes the application of traditional safety analysis and certification techniques impractical, as they usually require a complete specification of the system and its context in advance. In the following chapter, we introduce new techniques to meet this challenge and outline a safety certification concept specifically tailored to collaborative embedded systems.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_8

www.dbooks.org

8.1 Introduction and Motivation

Embedded systems are presently evolving from stand-alone closed embedded systems towards open and collaborative embedded systems (CESs). In CESs, collaboration can take place on different scales—from small, predefined groups of systems to large, heterogenous collectives of systems at a global level—and there will be an evolution from smaller scales to larger scales of collaboration [Damm et al. 2019]. Collaboration between CESs means sharing information concerning context perception, reasoning, and actuation. It means acting in unison with each other to reach a suoperordinate goal or to render a higher level service that could not be achieved by single systems alone.

The potential in this is very significant. For many existing applications, it will become possible to improve important performance properties (e.g., speed or efficiency), decrease resource consumption (e.g., fuel consumption), and improve safety (e.g., in a platooning scenario) simultaneously. The implications for society and the economy are correspondingly huge, as the systems envisioned have the potential to completely transform and improve our societies and economies in a groundbreaking way.

There are, however, a number of challenges that must be tackled before this vast potential can be unlocked. Most importantly, established engineering approaches—and safety engineering approaches and standards in particular—focus on closed systems and cannot be applied to future CESs without further ado.

Traditional safety engineering approaches typically require a complete understanding and specification of the system (and its context) under consideration at design time. CESs, however, may be integrated with other CESs at runtime and thus form collaborative system groups (CSGs) dynamically. Some of these systems may not even be on the market during the design phase of others. Therefore, the goal of obtaining a full specification of all possible variations of CSGs is generally not feasible at design time, and new safety certification approaches are required.

CrESt set out to investigate corresponding solution ideas based on a range of complementary industrial use cases. A key premise of our proposed safety assurance approach is that the required information is only partially available at design time and must be completed at runtime. This assumption means that certain parts of the assessment

process that traditionally take place at design time have to be postponed until runtime, when all variables can be resolved. This applies in particular to the final verdict as to whether the integrated CSG is safe or not. Nevertheless, we consider it essential to conduct as much preparatory work as possible during the design phase to ensure that the final assessment at runtime can be performed efficiently and in a largely automated way.

The remainder of this chapter is structured as follows: Section 8.2 gives a brief overview of the proposed safety certification process. Section 8.3 describes the integration of modular safety cases at runtime to obtain a coherent safety case for the systems group. Section 8.4 addresses the inner details of the modular contracts at different levels and how they can be standardized. Finally, Section 8.5 concludes this chapter with a summary.

8.2 Overview of the Proposed Safety Certification Concept

Our safety certification concept stipulates a two-stage process. The first step concerns the preparatory work at design time. Here, each CES is equipped with a modular safety case. In each modular safety case, the respective CES is conceived as a stand-alone system and an interface for the integration with other modular safety cases is defined. The main purpose of these modular safety cases is to provide safety arguments and evidence to enable better decision-making at runtime. To this end, modular safety cases include the working conditions of their respective CESs, such as requirements that are placed on the environment or other CESs. Furthermore, they specify guarantees that may or may not be given by the CES for its own services provided, depending on whether certain conditions are met or not. Hazard and risk analysis make it easier to understand potential hazards that may originate from a system. Context modeling and analysis allow identification of the uncertainties that may arise in a particular context, as well as shared resources that have to be coordinated between systems. Fault analysis investigates possible malfunctions and drives the definition of safety measures. Finally, all these aspects are then combined into a modular safety case for each system.

The second step in our safety certification concept concerns the integration of the modular safety cases at runtime with respect to the planned collaboration. For this purpose, information on how the CESs

www.dbooks.org

are organized among themselves and what types of dependencies emerge as a result is required. This information can, for example, originate from a human integrator who arranges the CESs (adaptable factory) or it can be generated by a machine — for example, from onboard computers (vehicle platooning). The resulting system architecture therefore reflects the interdependencies between the CESs, which are needed to integrate their modular safety cases. Finally, the integrated safety case is evaluated and used to assess the safety of the CSG as a whole.

To perform the second step, both semi-automated approaches with human intervention (cf. Section 8.3) and fully automated approaches (cf. Section 8.4) are feasible, depending on the context. In the case of a semi-automated approach, a human operator can be kept in the loop through manual selection of a suitable configuration, for example. In the case of full automation, the systems can negotiate their relevant properties on a peer-to-peer basis. This can be done on a contract basis by providing and demanding guarantees for services exchanged. At runtime, the safety of the collaboration is continuously monitored in a feedback loop. If the conditions for safe cooperation are no longer met, the systems must react accordingly — for example, by graceful degradation or termination of cooperation.

8.3 Assuring Runtime Safety Based on Modular Safety Cases

As already motivated, CESs must react to ongoing, constant changes in their open, dynamic context. However, for the systems to react at runtime, the CESs must first be aware of their relevant context [Petrovska and Grigoleit 2018] so that they can subsequently monitor and assess the type and impact of the context change. Additionally, in parallel, uncertainties resulting from these changes have to be handled effectively. To allow an efficient certification for the CSG at runtime, a dynamic risk assessment is performed. The systematic documentation of all relevant evidence enabled by modular safety cases helps safety engineers during the certification process.

In the context of an adaptable factory, major trends such as the growing individualization of products and volatility of product mixes lead to a situation where every product is produced differently and is routed according to the current production situation [Koren 1999], [Yilmaz 1987]. In this chapter, we demonstrate our methods using a small case study as a running example. Reconfigurable industrial CESs

(such as a robot arm and a tray as a storage unit) are used to assemble a small roll that consists of a roll body, an axle, and two metal discs as depicted in Figure 8-1.

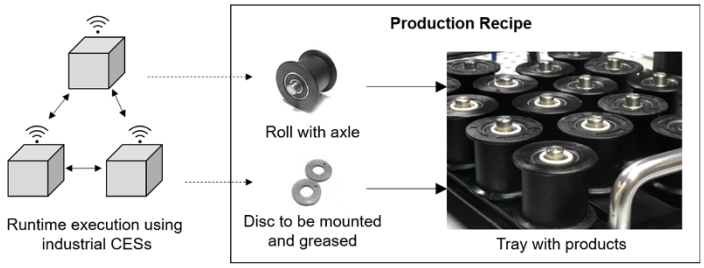


Fig. 8-1: *Case study description for an adaptable factory*

8.3.1 Modeling CESs and their Context

In practice, CESs are typically developed either within one original equipment manufacturer or by different suppliers. Moreover, when CESs form collaborative system groups (CSGs), the CSGs can hardly be analyzed a priori as relevant context because they are typically not explicitly defined at design time. On the contrary, they are formed, at least to a certain extent, emergently at runtime, which is actually a key trait and strength of CESs and the open ecosystems they enable. A CSG fulfills a global goal that an individual CES cannot fulfill alone. Of course, as already motivated, the increased complexity of the functionality requires different verification, validation, and certification approaches.

One method for testing a CES for consistency and correctness is the use of executable models, referred to as monitors. A monitor observes the execution of a system and determines whether it meets the given requirements [Goodloe and Pike 2010]. The monitor can then register and log the violations found during the test. In particular, in CSGs, monitors may help in detecting specification violations when the requirements are described as goals. One of the main characteristics of these goals is that they are influenced by the orchestration of the different CESs. However, for the systems to react at runtime, the CESs must first be aware of their relevant context through the constitution of runtime models of the context where the CESs operate, so that they can subsequently monitor and assess the type and impact of context changes on the systems (this is explained further in Section 8.3.3).

www.dbooks.org

Modeling the Context

Context awareness is generally accomplished through the creation of context models, which depict relevant aspects of the context for the CES. The context models are initially created at design time of the systems, and updated accordingly during runtime. However, in practice, there are differences in the context modeling concepts among different manufacturers and suppliers. This exacerbates the integration effort of different data models to integrate CESs, causing a "semantic heterogeneity" [Jirkovský et al. 2016]. The use of ontologies unfolds the potential to serve as a conceptual as well as a technological representation of such data models to cope with the semantic heterogeneity and enable semantic interoperability [Negri et al. 2016].

Content Ontology

We propose an ontology, shown in Figure 8-2, that integrates elements of two types of context: (1) classes and relationships of the interacting CESs, known as the operational context, and (2) sources of information with respect to the CESs, seen as the context of knowledge.

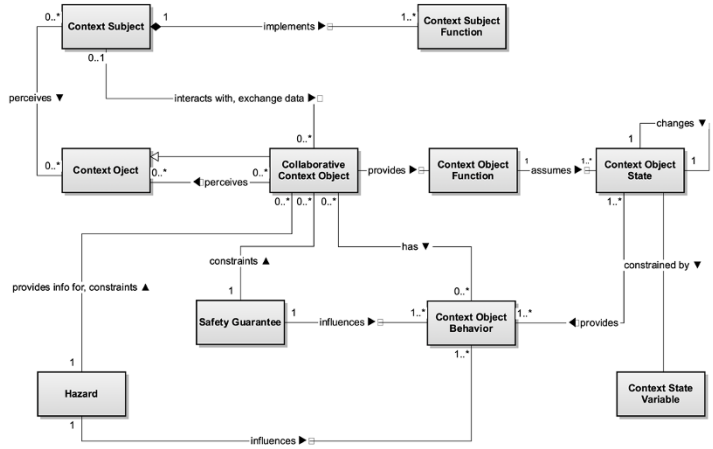


Fig. 8-2: *Context ontology*

The operational context models the interaction between a system under analysis and other systems in the environment, whereas the context of knowledge focuses on relevant knowledge sources that possess information about the system under analysis [Daun et al. 2016]. The aim of integrating these two types of context together is to

gather relevant classes for constructing a context model that includes relevant information that can subsequently be used to check specification violations during runtime monitoring.

Our proposed ontology allows a distinction between the system under analysis and the parts of the context that may influence the system but that cannot be changed. The system under consideration/analysis is called the "context subject." The context, composed of context objects, is that part of the environment that is relevant to the context subject. To allow the distinction between parts of the context that are collaborative and context objects that do not collaborate, we name a context object "collaborative context object." This distinction permits the identification of the entities that interact with the context subject, their dependencies with the context subject, and dependencies among context objects.

From a functional perspective, collaborative context objects provide services or functions that are accessible to the context subject. In our ontology, context object function entities are used to document the dependencies and the exchange of data between the context subject and these context functions. From a behavioral perspective, to enrich the documentation of a context function, we use context object state and context state variable entities. These entities provide information about the different states, and their related variables, that define the behavior of a context function. Furthermore, these context states define the context object behavior of collaborative objects in the context.

In the context of knowledge, the ontology integrates entities that provide and/or constrain the collaborative objects in the context. In particular, we are interested in safety guarantees and hazards that provide information and constrain context objects and the context subject based on standard rules.

Modeling Context in the Adaptable Factory

The creation of a context model is a process that is executed at both design time and runtime. At design time, the functional, structural, and behavioral aspects of the operational context of the CESs are modeled into a generic context ontology. This generic ontology can then be refined to create a domain-specific ontology that captures all relevant information of the domain — in our case, the adaptable factory use case. Both ontologies are represented as OWL files. The resulting ontology enables CESs to store context-related information and draw conclusions from this information. The domain-specific ontology of the adaptable factory serves two purposes: the ontology

www.dbooks.org

(1) defines the input data, defining in particular where the data is located and what the relationships between different data are. For a mechatronic object, for instance, the ontology may specify where this CES is located (i.e., its position in the machine cell). This results in purpose (2), where the ontology can be used to find constraints in the data. A mechatronic object may specify, for instance, the maximum speed at which the CES moves.

At runtime, the CESs are identified and a CSG configuration is selected. This information is then replicated into the adaptable factory context ontology: for each CES identified, a new individual (i.e., the instance of an entity in the ontology) is created and all the relevant information is stored as data properties of the new individual. Finally, the information stored in the context ontology is queried to build a runtime context model. For implementation and evaluation purposes, the runtime context model is stored in an XML file.

8.3.2 Runtime Uncertainty Handling

The term "uncertainty" is used in different ways in different research fields. Uncertainty and its impact are being extensively explored by research communities in areas such as economics, software and systems engineering, robotics, artificial intelligence, etc. In the field of cyber-physical systems, there are multiple definitions for uncertainty, and the one provided by [Ramirez et al. 2012], serves as a rationale for our collaborative systems: "Uncertainty is a system state of incomplete or inconsistent knowledge such that it is not possible for the system to know which of two or more alternative states/configurations/values is true." As explained, CESs interact and integrate at runtime; the uncertainties that occur during runtime, more specifically the ones that might create safety-critical scenarios for CSGs, are of prime importance here.

Two types of uncertainties can be distinguished: epistemic uncertainties and aleatory uncertainties [Perez-Palacin and Mirandola 2014]. The epistemic type refers to the uncertainties that arise due to incomplete knowledge or data, and the aleatory type of uncertainty is the result of the randomness of certain events. Epistemic uncertainties can be handled effectively by collecting additional information, meaning that the uncertainty then ceases to exist. In contrast, aleatory uncertainties are relatively complicated because of their inherent randomness. The concept presented will help to address most of the epistemic types of uncertainties and few

of the aleatory type. The main viewpoint of handling uncertainties explained in this section would be from that of safety assurance.

Concept Overview

The outline of the concept is to provide a quantified, well-reasoned, and well-defined mapping of the uncertainties identified to their corresponding mitigation steps. The CSG is constantly monitored at runtime for occurrences of uncertainty and, based on the definitions and parameters of these occurrences, runtime adaptations of configurations for CESs or any further specific measures defined in the mapping are undertaken to ensure safety.

Development of a U-Map for the Adaptable Factory

The solution approach is centered around the development of an uncertainty map (U-Map) artifact during design time. This artifact is used as the knowledge base at runtime for monitoring and executing mapped mitigation measures for uncertainty occurrences. The first step in the development of a U-Map is identifying the relevant uncertainty and its classification. This step is the most vital and also the most time-consuming. Here, all possible uncertainties are listed based on various classifications from research, the most recent and extensive being the one from [Cámara et al. 2015]. To aid the process of identifying uncertainties with respect to the information exchange between CESs from an ontological perspective, the classification provided by [Hildebrandt et al. 2019] is used. Both of these classifications are used as a checklist to identify possible uncertainties at runtime, specific to the use case. Once identified, concrete instances of uncertainty must be defined. In due process, uncertainties that can be resolved during the design of the CSG but have not been considered in general system development have to be updated. These instances have to be further iterated and quantified as monitor specifications so that they can be detected at runtime. Examples include ambiguity in sensor measurements, inconsistency in service descriptions, incompleteness in self-descriptions of CESs, or incompleteness in information exchange. The next step involves identifying all possible failures that might occur from these uncertainties that might put the system into a hazard state and might subsequently lead to an accident or harm. To aid this, standardized hazards and failures from [ISO 2010] are considered for the adaptable factory and from [ISO 2018] for vehicle platooning. Bayesian networks [Halpern 2017] and the Dempster-Shafer theory [Shafer 1976] based on probability theory are found to be effective for mapping the uncertainties identified to

www.dbooks.org

possible failures and hazards. As an outcome, we notice that each uncertainty can lead to multiple hazards and every hazard can be a result of one or more uncertainty occurrences. The next step involves mapping these hazards to their corresponding mitigation measures. For the use case of the adaptable factory, an intermittent step of rectification acts as an additional layer of safety assurance, which is feasible because of the semi-automated approach employed. The uncertainties that can be eliminated by rectification measures occur predominantly in information exchange between individual CESs. In certain cases, the system may still be in a hazardous state even after the uncertainty has been eliminated through rectification. To maintain safety, these hazards must be further mapped to appropriate mitigation measures. The mitigation measures can be either based on present industrial standards or they can be reconfiguration identified as degradation modes. In certain scenarios, these degradation modes alone are not sufficient and additional protective measures have to be taken.

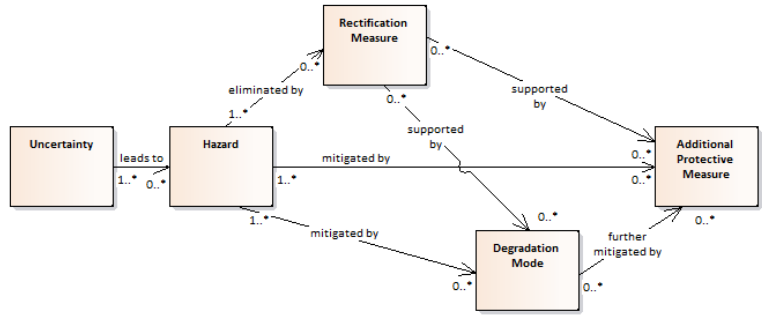


Fig. 8-3: *Visualization of a U-Map*

In the end, an extensive set of identified uncertainties is mapped to an even bigger set of possible hazards, which in turn is mapped to a rather small set of degradation modes and protective measures. This U-Map makes implementation simple and does not have an exploded range of mitigation measures that have to be undertaken specifically to handle every uncertainty. However, creating such a map and ensuring its completeness to handle all possible uncertainties at runtime can be a complex task, which presently relies greatly upon the research communities' identified sources of uncertainty, which in themselves might not be complete. Furthermore, we consider subjective probability for uncertainty occurrence [Shafer 1976],

which in itself might be imprecise. A U-Map can be visualized as shown in Figure 8-3.

At runtime, with the help of this U-Map, the necessary rectification measures are taken by the safety engineers, thereby eliminating relevant uncertainties before safety approval. The degradation modes and additional protective measures serve as an input for further explanation of dynamic safety certification, in that they enable the appropriate configuration and safety measures to be chosen.

8.3.3 Runtime Monitoring of CESs and their Context

The generated runtime context model from Section 8.3.1 can be used to deliver relevant information that enables runtime analysis/monitoring.

With the information available from the context model explained in Section 8.3.1 and the specifications for uncertainty detection from the DTU map as explained in Section 8.3.2, monitors can be created to monitor the properties of interest in a given CES. The monitors and the specifications are created at design time; however, the monitors are executed during runtime. For example, it may be desirable to monitor the speed of a mechatronic object to determine whether the said speed obeys safety requirements. A common way to create a runtime monitor is to translate assertions about the state of a context element into rigorous specification formalisms [Bartocci et al. 2018], such as LTL formulas, to subsequently create instrumentation files with the monitor specifications. In our example, a domain expert can provide the assertion "It is always the case that CES1 moves at a speed of 2 mm/s" and this can be translated into the LTL formula $G(ces1.speed \leq 2)$; this formula can be used to create the monitor specification [Bartocci et al. 2018] as instrumentation files[1] that have to be integrated into the CES. The runtime monitor specification must be created during design time and the instrumentation files generated should be integrated during development. At runtime, these monitor specifications, including the specifications from the DTU map, will be represented in the form of modular safety cases. In the context of an adaptable factory, a centralized software that is responsible for task orchestration and system assessment can identify and compile the monitoring requirements dynamically to allow for the final approval by safety engineers in a semi-automated certification process.

[1] http://fsl.cs.illinois.edu/index.php/MOP

www.dbooks.org

8.3.4 Integrated Model-Based Risk Assessment

Due to frequent changes in the products being manufactured, adjusting a factory quickly is a major challenge. This raises concerns with regard to dependability due to unknown configurations at runtime. Thus, apart from functional aspects (i.e., the check of whether a factory is able to manufacture a specific product), safety aspects as well as product quality assurance aspects must be addressed. In flexible production scenarios, a risk assessment must be conducted after each reconfiguration of the production system. Since this is a prerequisite for operating the factory in the new configuration, a manual approach can no longer effectively fulfil the objectives for assuring safety in highly flexible manufacturing scenarios. During production, every process step has the potential to influence the quality of the product in an undesirable way, for example depending on the precision of the equipment used, or random failures while executing the process step. This is captured in a Process Failure Mode and Effects Analysis (process FMEA) with the concept of failure modes of a process step as well as the respective severity. The process FMEA also defines measures for detecting and dealing with unwanted effects on product quality. Since both the factory's configuration and its products change constantly in adaptable factory scenarios, a process FMEA must be performed dynamically during operation.

In the context of industrial production systems, the safety standards ISO 13849 [ISO 2006] or IEC 62061 [IEC 2005] provide guidelines for keeping the residual risks in machine operation within tolerable limits. For every production system, a comprehensive risk assessment is required, which includes risk reduction measures if necessary (e.g., by introducing specific risk protective measures such as fences). The resulting safety documentation describes the assessment principles and the resulting measures that are implemented to minimize hazards. This documentation lays the foundation for the safe operation of a machine and it proves compliance with the Machinery Directive 2006/42/EC of the European Commission [European 2006].

In this section, we present an approach for the model-based assessment of flexible and reconfigurable manufacturing systems based on a meta-model. This integrated approach captures all information needed to conduct both risk assessment and process FMEA dynamically during the runtime of the manufacturing system in an automated way. The approach thus enables flexible manufacturing

scenarios with frequent changes in the production system up to a lot size of one.

Meta-model SQUADfps

To address the aforementioned problem statement for a dynamic assessment at runtime, a meta-model called SQUADfps (machine Safety and product QUAlity assessment for a flexible proDuction system) is presented [Koo, Rothbauer et al. 2019]. This metamodel considers hazards and failure modes due to both safety and quality issues. Four categories are introduced within the SQUADfps metamodel: *process definition*, *abstract services*, *production equipment*, and *process implementation*. This depicts the modularity within an adaptable factory scenario. This integrated model-based approach allows information not only from each item of modular production equipment (i.e., CESs within CrESt) to be considered during the assessment, but also from the production context.

With the focus on quality assurance, an integrated CES that provides services for production (EquipmentService) steps brings along information about its possible failure modes (EquipmentFailureMode) at runtime. Equipment that provides quality measures (CoveredFailureMode) brings along the information about the effectiveness of the measures (e.g., detection) regarding specific failure modes (EquipmentFailureMode). The suitability of the planned production schedule—that is, the equipment's suitability to provide the required services—can be analyzed by conducting a model-based quality assessment process FMEA, taking the production recipe and the services required into account, as shown in Figure 8-4. For the risk assessment, possible hazards introduced into the overall production system during process implementation can be captured and checked against the available SafetyFunction to determine whether safety requirements are fulfilled.

The benefits of applying SQUADfps for the dynamic certification of CSGs in an adaptable factory are twofold: firstly, this metamodel allows risk-related information to be captured dynamically at runtime. Secondly, the risk information—be it hazards or failure modes along with the analysis of this information—provides input for the modular safety cases systematically. The process of conducting a dynamic safety certification is discussed in subsequent paragraphs.

www.dbooks.org

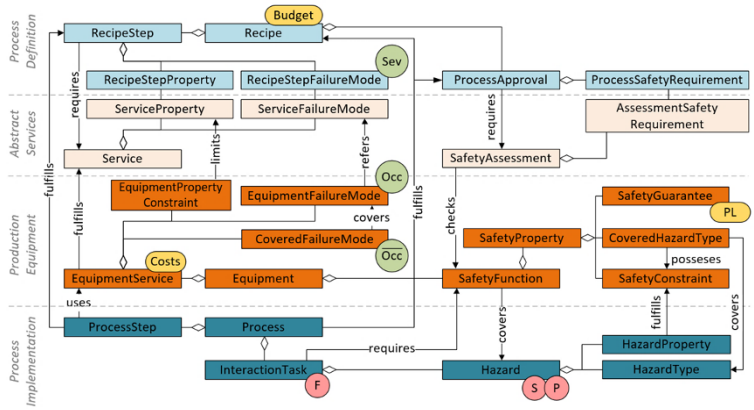


Fig. 8-4: *Meta-model SQUADfps for a dynamic machine safety and product quality assessment at runtime [Koo, Rothbauer et al. 2019]*

Based on the case study described, we now present the results generated using SQUADfps to aid understanding.

Case Study Example

Table 8-5 shows the product recipe R = r_1, …, r_6 for producing a pulley wheel, specifying the required process steps. For each recipe step required, the relevant failure modes are listed and a measure of their severity (Sev) is given as they would impact the final product. This information can be added by the design team of the product, as they know exactly how each failure mode will impact the final product. For each failure mode in the product recipe, a measure of the detectability (Det) by scheduled quality measures is also given. For example, the failure mode Misplacement of the service Pick & place can be detected visually with high certainty (detection value is one) but the failure mode Crimping will likely go unnoticed as these can only be detected by stress tests that are not considered in this process instantiation.

Consider the first process instantiation P in Table 8-5, consisting of process steps p_1, …, p_{6a}. Process P is capable of producing the pulley wheel, as it provides all the services required and in the correct order. For each deployment of a required recipe step to a process step on a concrete item of equipment, the occurrence—the information regarding failure mode frequency (Occ)—can be added to the model. This information is provided by the vendor of the production equipment that provides production services and the operator will

possibly also update these values based on local production experience (e.g., environment conditions).

Recipe R	Failure Mode	Sev	Service	Occ	Process P	Det	RPN	Occ'	Process P'	Det'	RPN'
r_1: Deliver tray	Misplacement	4	Convey	2	p_1: Belt conveyer	1	8	2	p'_1: Belt conveyer	1	8
	Shock	5		1		1	5	1		1	5
r_2: Mount axle	Misplacement	4	Pick & place	2	p_2: Robot arm	1	8	2	p'_2: Robot arm 2	1	8
	Crimping	2		4		5	40	2		2	8
r_3: Circular grease roll	Too little	5	Apply liquid	4	p_3: Robot arm	1	20	4	p'_3: Robot arm 2	1	20
	Too much	2		4		1	8	4		1	8
r_4: Mount 1st disc	Misplacement	4	Pick & place	2	p_4: Robot arm	1	8	2	p'_4: Robot arm 2	1	8
	Crimping	3		4		5	60	2		2	12
r_5: Circular grease 2nd disc	Too little	5	Apply liquid	4	p_5: Robot arm	1	20	4	p'_5: Robot arm 2	1	20
	Too much	2		4		1	8	4		1	8
r_6: Mount roll on 2nd disc	Misplacement	4	Pick & place	2	p_6: Robot arm	1	8	2	p'_6: Robot arm 2	1	8
	Crimping	5		4		5	100	2		2	20
			Visual inspection		p_{6a}: Worker				p'_{6a}: Laser scanner		

Table 8-5: *Case study process definition and two possible deployments of the Process P and P' (production schedules)*

Looking at the risk priority numbers (RPN), the chosen process deployment P for producing the product seems to come at a high risk of not reaching the required quality goals, which is indicated by the high value of the RPN. An alternate process instantiation using more reliable equipment and higher precision quality measures can be seen in Process P' in Table 8-1. The equipment Robot arm 2 has a lower probability of introducing the critical crimping failure mode (Occ value 2 for Robot arm 2 vs. Occ value 4 for Robot arm 1) and a high-precision laser scanner is used as a quality measure. As we can see, the concrete instantiation of the process on actual equipment influences the occurrence values for each failure mode of a production step as well as detection values. As a consequence, the risk measured—for example, using the product of occurrence and severity (RPN)—will differ and the highest values of RPN are lowered from 100 to 20.

Considering machine safety, the results generated using model-based risk assessment for the various integrated CESs can be seen in Table 8-6 (for the production schedule P'). In this table, the combination of the risk parameters F (Frequency), S (Severity) and P (Possibility for avoidance) will determine the risk level (which is represented as the Performance Level PL used for safety analysis).

www.dbooks.org

Process P'	Interaction Task T	F	Hazard H	S	P	PL'	Safety Function	PL
Robot arm 2	Loading of roll body with axle	F2 (high)	h_1: Shearing due to robot movement	S2 (high)	P2 (high)	PL e	Safety-sensitive cover	PL e
Belt conveyor	Loading of roll body with axle	F2 (high)	h_2: Squeezing due to belt getting caught	S1 (low)	P1 (low)	PL b	Light curtain	PL d
Robot arm 2	Maintenance of robot's handling tool	F1 (low)	h_3: Bruising due to robot movement	S1 (low)	P1 (low)	PL a	Safety-sensitive cover	PL e

Table 8-6: *Exemplary results for the model-based risk assessment*

In the exemplary safety risk assessment shown, we can see that the integrated robot (CES) might cause a hazard h_1 shearing when the operator loads the material into the assembly cell. The runtime assessment system evaluates this risk as PL e (very high risk according to ISO 13849-1) based on different data from the context and allocates a possible existing safety function to h_1. As the integrated safety-sensitive cover for the robot has a very high reliability (also PL e), it provides proof that the risk of h_1 can be mitigated during the interaction task. A similar analysis procedure is also performed for all relevant hazards to generate the foundation for the safety risk assessment.

This approach is of a qualitative nature, which in practice is very effective for prioritizing measures for the main problems. It can be extended to deliver quantitative measures of production risk. The approach aims to assist humans in finding an optimal solution for producing a product while considering both machine safety and product quality aspects.

8.3.5 Dynamic Safety Certification

The goal of a dynamic and runtime safety certification in the context of an adaptable factory allows an accelerated operational safety approval (i.e., certification) after system modifications are performed. With the dynamic safety certification method presented, automated capture and analysis of runtime data can be performed more efficiently. In the production domain, the human's role as the person responsible remains significant to guarantee system safety in accordance with the European Machinery Directive [European 2006]. Therefore, a human-in-the-loop assurance based on the concept of modular safety cases [Kelly 2007] is proposed for the adaptable factory use case.

The concept of using a modular safety case allows relevant requirements (i.e., safety goals) and analysis results (i.e., argument and evidence) to be documented in a systematic way for the required certification process. The initiation of these modular safety cases highlights the context-relevant requirements that must be fulfilled by the specific runtime system configuration—as already mentioned earlier—to deal with monitoring, uncertainty, and risk requirements. Dealing with all these requirements successfully and the completion of modular safety cases at runtime will contribute to the overall certification of the adapted CSG.

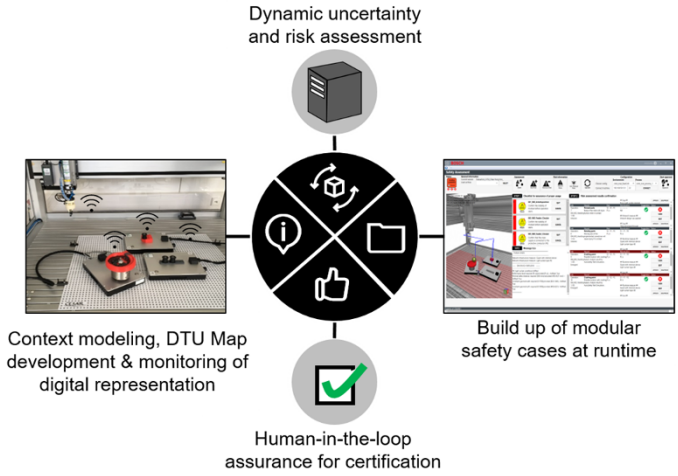


Fig. 8-7: *Dynamic safety certification for adaptable factory*

An interactive tool called AutoSafety [Koo, Vorderer et al. 2019] has been developed to help operators and safety engineers to assess and approve the adaptable assembly demonstrator at runtime (the dynamic safety certification process is shown in Figure 8-7). This semi-automated certification approach builds up the safety case of the CSG by integrating modular safety cases of the integrated modular systems while considering relevant runtime safety aspects (e.g., runtime measures) identified during reconfiguration. Moreover, AutoSafety will be able to highlight the status of each modular safety case individually with regard to whether they are successfully fulfilled based on runtime data. When automated analyses of certain runtime variables are conducted, the respective modular safety cases can be updated automatically. Humans can also perform updates to ensure the correctness, accuracy, and completeness of the results. For highly adaptable factory scenarios in the future, this dynamic runtime

www.dbooks.org

certification approach will be able to accelerate the safety approval procedure and minimize manual engineering efforts required for assessment and documentation.

8.4 Design and Runtime Contracts

This part of the chapter explains the use of contracts within the specification, development, and standardization of safety-critical collaborative systems. The concepts are illustrated in connection with the use case "Vehicle Platooning"

One of the biggest challenges with collaborative systems is to ensure that the systems behave safely — not only as individuals but also as an integrated system. At the same time, a collaborative system can only be successfully introduced into the market if its safety can be assured — for example, based on an adequate certification process. This could be achieved as part of the design time engineering or based on a combination of design time engineering and runtime assurance/certification measures. In the first case (i.e., safety assurance achieved during design time engineering), a traditional method of system development would be pursued, with the difference being that this would be done for the integrated system, which is an abstract construct. However, this requires that the CESs and CSGs are known to a sufficient extent — for example, by means of comprehensive standardization of system and service characteristics for a domain. The second case (i.e., safety assurance based on a combination of design time engineering and runtime assurance/certification measures) is ideal, given the natural dynamics of collaborative systems. To achieve this, we require a fully integrated and comprehensive solution (e.g., runtime certification, an infrastructure for communication, certain standardization — for example, with regard to interoperability, tracking, evaluation, enforcement, etc.). However, it is impossible to decide what this solution should look like for future systems, although we can make pragmatic assumptions as far as we require these aspects for our work. In this project, we have focused in particular on closing the gap between traditional design time certification and runtime certification. We have done this by introducing an approach for collaborative systems specification that relies heavily on contract-based design and engineers the exchange of guarantees and demands/assumptions during runtime. During the design phase, contracts allow the distribution of responsibilities of the participants

to be defined, and during runtime, they allow safe behavior to be enforced.

8.4.1 Design-Time Approach for Collaborative Systems

One of the main drivers in the definition of our approach is the lack of understanding of how to establish a safe collaboration. Therefore, it is not our aim to find the best solution for a particular aspect. Instead, we are aiming for a more comprehensive solution that could help us to better understand the problem and thus distinguish and highlight the more important aspects when considering certification for safe behavior. For this reason, the approach defines the need to specify and certify the CSG itself. The goal is to make the CSG specification the standard that defines the minimum requirements for collaboration in a specific scenario (in our case, a vehicle platoon). System developers who want to participate in such collaboration must then comply with the specification and the associated domain regulations.

Creating the CSG Specification

To build a CSG specification systematically, we consider the following refinement steps.

At the business/domain level, the CSG designer must initially define the aim or subject matter of the collaboration. We believe that, given the nature of collaborative systems, service-oriented architectures (SOA) [Bell 2008] offer useful concepts for specifying this aspect. These include the specification of the functions and objectives of collaboration, roles, allowable system compositions, structural configurations, environmental constraints, and the definition of service contracts.

At the functional level, and by following a traditional top-down approach, the reference architecture that defines how functionality and responsibilities are distributed among roles is built. This includes defining the minimum requirements, the behavior, and functions of the roles and their dependencies, and setting the flexibility points. As mentioned above, runtime contracts could be used to enable such flexibility points. We consider ConSerts [Schneider 2013] to be a useful technique for realizing this concept since they are contracts that are specifically designed to be exchanged during runtime. ConSerts include concepts for defining the quality of the data to be exchanged, and they can be used to define the reactions to contract violations and discrepancies that will guide the change of behavior in the system.

www.dbooks.org

At the contract level, the design decisions have been refined enough so that the CSG designer can define the final list of requirements in the form of verifiable contracts. This should be done in a more formal way to avoid misinterpretations by the CES developers.

Safety-Relevant Activities

In parallel to the design, the safety activities are performed:

At the business/domain level, the safety engineer should be able to perform the hazard and risk analysis for the CSG thanks to an initial list of system functions. This takes place at two levels:

At the CSG level, the consequences of the fail behavior of functions at CSG level must be investigated — for example, platoon deceleration. A lack in deceleration of a platoon can lead to a mass collision. This clearly has a higher severity than a single vehicle collision.

At the role level, the fail behavior of the subsystems in the collaboration, according to the initial function distribution among the roles, must be investigated with regard to its effect on the integrated system.

At the functional level, and given the specification of the safety goals and a first draft of the functional architecture, fault analysis can be performed in the form of a Component Fault Tree (CFT) [Domis and Trapp 2009]. This allows safety measures to be identified and the current design to be adapted to avoid or mitigate these failures. Safety measures are represented by safety requirements, which can be mapped directly to the collaboration roles. With a safety strategy in mind and the design that reflects it, it is then possible to create the functional safety concept for the CSG level.

At the contract level, and similar to the procedure for the architecture design, the safety requirements mentioned must be defined in terms of verifiable safety contracts.

8.4.2 Contracts Concept

As mentioned above, the CSG specification defines the functionality and behavior that the roles will take on in a collaboration. This is partly defined by functional and safety contracts. These contracts are considered as pure design-time contracts since they are exchanged and consumed only during the CSG-CES development time. On the other hand, ConSerts should be exchanged during runtime. In this approach, this means that ConSerts must also be developed and be

standardized as part of the CSG specification so that they can also be successfully exchanged and consumed at runtime.

In the context of the vehicle platoon use case:

- Functional contracts were primarily defined based on the state machine of each role. They define the behavior relevant for collaboration in a particular state.
- Safety contracts define the reaction to failure situations. Therefore, they mainly refer to the transitions in the state machine that connect normal states and failed operational states (including degraded states).
- ConSerts were engineered as an additional function of the system in close relation to the service contracts defined in the context of the service-oriented architecture.
- Service contracts define the specific messages exchanged between leader and follower. Therefore, ConSerts were defined in the form of guarantees of the safety-relevant data being exchanged.
- ConSerts are consumed according to the reference architecture for three purposes: to support flexibility, to allow valid CSG compositions, and to drive change of states.

Flexibility aspect: If demands and guarantees are met, the collaboration is allowed. Flexibility is supported because demands define a range in which guarantees can satisfy them. If the guarantees remain within this acceptable range, collaboration is allowed.

Valid compositions: A valid composition means that a demand is satisfied by a specific guarantee. If this is not the case, the collaboration should theoretically be terminated. We consider the validation of demands vs guarantees in two ways:

Contract violation: A violation is deemed to have occurred when the vehicle with demands can prove on its own that the service provider is not acting in accordance with its guarantees.

Contract discrepancy: A discrepancy arises when a demand cannot be satisfied by any guarantee.

Change of states: A contract violation is engineered in the platoon scenario such that when it occurs, the vehicle that detects the violation will preventively transition into a degraded mode for a certain time and notify the system causing the problem. In the event of a contract discrepancy, the collaboration with the provider is terminated. This will finally lead to the division of the platoon into sub-platoons.

www.dbooks.org

8.4.3 Runtime Evaluation of Safety Contracts

A full detailed runtime analysis and safety assurance of all collaboration scenarios, including all environmental conditions, is not possible for real systems. Functional and safety contracts provide the means to operate on an adequate abstraction that has been prepared by diligent development time engineering. The use of safety contracts of different CESs requires the development of an environment capable of composing and evaluating these contracts at runtime. In the vehicle platoon use case investigated, safety contracts are used to define the reaction to failure situations, and safety guarantees are expressed as a means for tolerating deviation from a nominal behavior.

Simulative Approach for Validation of Safety Contracts

In order to validate the safety contracts designed and evaluate the behavior of the overall system when failures occur, a simulative approach can be used. Simulations and model-based evaluation of safety contracts are used during the development phase to observe the system behavior and validate the expectation of the safety engineer at design time. In the simulation, various manipulations, such as data corruption, invalid data due to a hardware defect, and other possible failures can be injected into the system [Isermann 2017]. An executable model of the collaborative embedded system should be created first as a means of validating the required safety functionalities.

Safety contracts separate requirements into assumptions and guarantees, which help to decrease the complexity of verifying the implementation against its specifications. Using a formal approach such as failure detection and isolation (failure handling) to do this allows the process of contract evaluation to be automated.

Case Study: Vehicle Platoon Example

The aim of the vehicle platoon use case is due to maintaining a short inter-vehicle distance. This would be achieved by exploiting real-time knowledge of the driving behavior of each vehicle in the platoon through onboard sensors and wireless communication among platoon members. If a sensor or communication failure occurs, or the respective safety guarantees become worse due to context changes, then the real-time knowledge would not be reliable, which puts the platoon in an unsafe mode. Therefore, both failures and changes in safety guarantees must be detected and compensated to keep the system working under any circumstance. Using a graceful degradation concept would help the system to remain operational (with a

degraded performance) in at least some such conditions. Note that the simulative approach used in the CrESt project is not executed in a fully realistic scenario due to effort limitations; instead, a highly simplified scenario has been used. The simulation model focuses on a platoon system that is already running, consisting of three vehicles running on one straight highway without any tunnels, curves, or inclines. In the simulation runs, one predefined safety contract is evaluated as an example. The results of the simulation are presented in Figure 8-8 and Figure 8-9. These figures show the system behavior in the event of a distance measurement sensor failure.

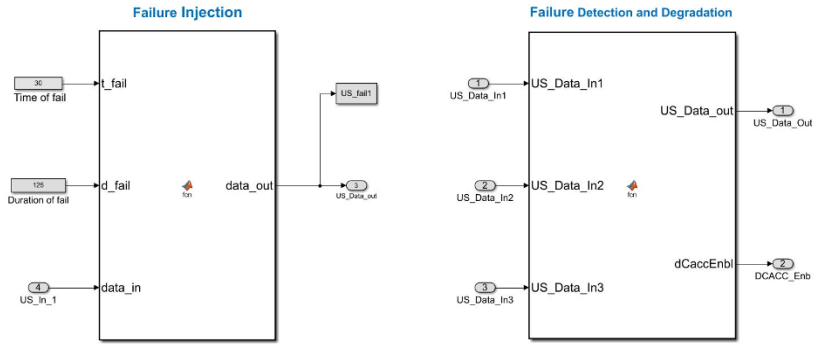


Fig. 8-8: *Example: Generation and handling of errors with Simulink*

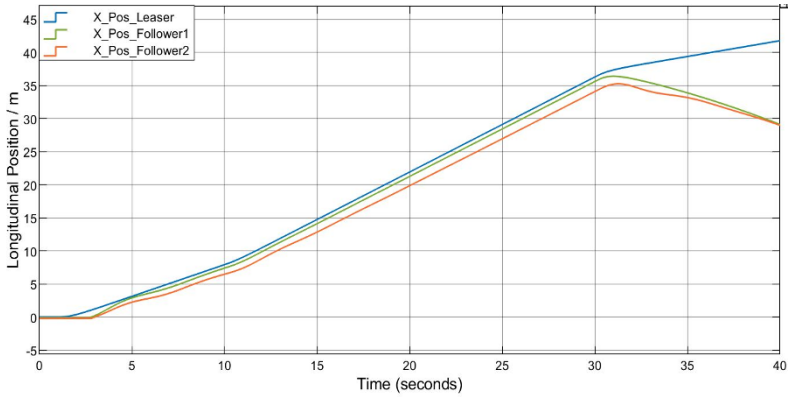


Fig. 8-9: *Example of error injection simulation results in a longitudinal platooning model using Simulink*

As shown above, the failure injection block (in the left side) in Figure 8-8 is implemented as a MATLAB function in Simulink and is located before the sensor inputs into the controller block. It can generate invalid sensor values at a specified time with the desired repetition rate of the error. Moreover, the failure detection and degradation

www.dbooks.org

function validates the incoming data before it is passed on to the controller. Figure 8-9 shows the course of the platooning without applying error detection and degradation. Here, it becomes obvious that a sensor defect causes a deviation in the platooning distances because of the impacted controller performance. The third vehicle from Figure 8-8 is still following the previous vehicle because it is receiving correct speed data.

8.5 Conclusion

In this chapter, we presented a concept for safety certification of collaborative embedded systems. We highlighted the most distinct characteristics that distinguish them from classical systems. It is mainly their dynamicity that makes predicting their behavior difficult and therefore renders traditional safety certification techniques impractical. Based on these considerations, we presented new techniques and adaptations of existing techniques to enable a safety certification process that is specifically tailored to collaborative embedded systems.

We have outlined a two-step process. On the one hand, this process comprises the preliminary work during the design phase. All CESs are equipped with modular cases that contain an interface for integration with other safety cases. Since there are still many unknowns during the design phase, the second part of the safety certification process is performed at runtime, when all variables can be resolved. At runtime, the modular safety cases are integrated and evaluated according to the planned collaboration. Our concept comprises the monitoring of context changes at runtime and facilitates the handling of uncertainties. This enables a largely automated process that can be repeated efficiently during dynamic reconfigurations at runtime.

8.6 Literature

[Bartocci et al. 2018] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, S. Sankaranarayanan: Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In: *Lecture Notes in Computer Science*, Springer, Cham. 2018, pp. 135-175.

[Bell 2008] M. Bell: Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley Publishing, 2008.

[Cámara et al. 2015] J. Cámara, G. Moreno, D. Garlan: Reasoning about Human Participation in Self-Adaptive Systems. In: 2015 IEEE/ACM 10th International

Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE, 2015, pp. 146-156.

[Damm et al. 2019] W. Damm, P. Heidl: Position paper on Safety, Security, and Certifiability of Future Man-Machine Systems, Results of the SafeTRANS Working Group "Resilient, Learning, and Evolutionary Systems", https://www.safetrans-de.org/en/activities/Roadmapping.php, 2019.

[Daun et al. 2016] M. Daun, J. Brings, T. Weyer, B. Tenbergen: Fostering Concurrent Engineering of Cyber-Physical Systems: A Proposal for an Ontological Context Framework. In: 2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), IEEE, 2016, pp. 5-10.

[Domis and Trapp 2009] D. Domis, M. Trapp: Component-Based Abstraction in Fault Tree Analysis. In: International Conference on Computer Safety, Reliability and Security, Springer-Verlag, 2009, pp. 297-310.

[European 2006] European Commission: Machinery Directive 2006/42/EC 2006. 2006.

[Goodloe and Pike 2010] A. Goodloe, L. Pike: Monitoring Distributed Real-Time Systems: A Survey and Future Directions. 2010.

[Halpern 2017] J. Y. Halpern: Reasoning about Uncertainty. MIT press, 2017.

[Hildebrandt et al. 2019] C. Hildebrandt, T. Bandyszak, A. Petrovska, N. Laxman, E. Cioroaica, S. Törsleff: EURECA: Epistemic Uncertainty Classification Scheme for Runtime Information Exchange in Collaborative System Groups. SICS Software-Intensive Cyber-Physical Systems, 34(4), (2019), pp. 177-190.

[IEC 2005] International Electrotechnical Commission (IEC): IEC 62061: Safety of machinery — Functional Safety of Safety-Related Electrical, Electronic and Programmable Electronic Control Systems. 2005.

[Isermann 2017] R. Isermann: Fahrerassistenzsysteme 2017: Von der Assistenz zum automatisierten Fahren — 3. Internationale ATZ-Fachtagung Automatisiertes Fahren. Springer-Verlag (available in German only).

[ISO 2006] International Organization for Standardization (ISO): ISO 13849–1: Safety of Machinery — Safety-Related Parts of Control Systems – Part 1: General Principles for Design. 2006.

[ISO 2010] International Organization for Standardization (ISO): ISO 12100: Safety of Machinery - General Principles for Design — Risk Assessment and Risk Reduction. 2010.

[ISO 2018] International Organization for Standardization (ISO): ISO 26262-3: Road Vehicles — Functional Safety — Part 3: Concept Phase. 2018.

[Jirkovský et al. 2016] V. Jirkovský, M. Obitko, V. Mařík: Understanding Data Heterogeneity in the Context of Cyber-Physical Systems Integration. In: IEEE Transactions on Industrial Informatics 13, no. 2, 2016, pp. 660-667.

[Kelly 2007] T. Kelly: Using Software Architecture Techniques to Support the Modular Certification of Safety-Critical Systems. In: ACM International Conference Proceeding Series, Vol. 248, 2007, pp. 53–65.

[Kephart and Chess 2003] J.O. Kephart, D.M. Chess: The Vision of Autonomic Computing. In: Computer 36, no. 1, 2003, pp. 41-50.

[Koo, Rothbauer et al. 2019] C. H. Koo, S. Rothbauer, M. Vorderer, K. Höfig, M. Zeller: SQUADfps: Integrated Model-Based Machine Safety and Product Quality for

www.dbooks.org

Flexible Production Systems. In: International Symposium on Model-Based Safety and Assessment, Springer, 2019, pp. 222–236.

[Koo, Vorderer et al. 2019] C. H. Koo, M. Vorderer, S. Schröck, J. Richter, A. Verl: Assistierte Risikobeurteilung für wandlungsfähige Plug-and-Produce Montagesysteme. In: VDI-Kongress Automation, 2019, pp. 41–54 (available in German only).

[Koren et al. 1999] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, A. Galip Ulsoy, H. van. Brussel: Reconfigurable Manufacturing Systems. CIRP Annals, Vol. 48, 1999, pp. 527–540.

[Negri et al. 2016] E. Negri, L. Fumagalli, M. Garetti, L. Tanca: Requirements and Languages for the Semantic Representation of Manufacturing Systems. In: Computers in Industry 81, 2018, pp. 55-66.

[Perez-Palacin and Mirandola 2014] D. Perez-Palacin, R. Mirandola: Uncertainties in the Modeling of Self-Adaptive Systems: A Taxonomy and an Example of Availability Evaluation. In: Proceedings of the 5th ACM/SPEC international conference on Performance engineering, 2014, pp. 3-14.

[Petrovska and Grigoleit 2018] A. Petrovska, F. Grigoleit: Towards Context Modeling for Dynamic Collaborative Embedded Systems in Open Context. In: MRC@ IJCAI, 2018, pp. 41-45.

[Ramirez et al. 2012] A. J. Ramirez, A. C. Jensen, B. H. Cheng: A Taxonomy of Uncertainty for Dynamically Adaptive Systems. In: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE. 2012, pp. 99-108.

[Schneider and Trapp 2013] D. Schneider, M. Trapp: Conditional Safety Certification of Open Adaptive Systems. In: ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2013.

[Shafer 1976] G. Shafer: A mathematical theory of evidence (Vol. 42). Princeton university press, 1976.

[Yilmaz and Davis 1987] O. S. Yilmaz, R. P. Davis: Flexible Manufacturing Systems: Characteristics and Assessment. In: Engineering Management International, Vol. 4, 1987, pp. 209–212.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Patricia Aluko Obe, University of Duisburg-Essen
Jennifer Brings, University of Duisburg-Essen
Marian Daun, University of Duisburg-Essen
Linda Feeken, Offis e.V.
Elham Mirzaei, InSystems Automation GmbH
Martin Neumann, InSystems Automation GmbH
Jochen Nickles, Siemens AG
Simon Rösel, Model Engineering Solutions GmbH
Markus Sauer, Siemens AG
Holger Schlingloff, Fraunhofer FOKUS
Ingo Stierand, Offis e.V.
Jan-Stefan Zernickel, InSystems Automation GmbH

9

Goal-Based Strategy Exploration

When collaborative embedded systems (CESs) connect to form a group, this collaborative system group (CSG) can achieve goals that are beyond the reach of individual systems. The goals such a group can achieve depend on the constituent collaborative embedded systems. Consequently, the ability of a collaborative system group to adapt itself is driven by the capabilities of its collaborative embedded systems. This tight interconnection impedes the manual handling of adaptation strategies. Therefore, this chapter introduces a goal-based approach for strategy exploration that considers the peculiarities of collaborative system groups and collaborative embedded systems. The chapter sets out the model-based approach to adaptive system (group) design, incorporating the goals of collaborative system groups and individual systems, and outlines corresponding automated validation methods. We demonstrate the applicability of our approach for a case example of collaborative transport robots.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_9

www.dbooks.org

9.1 Introduction

Challenges

The development of collaborative embedded systems (CESs) faces challenges due to the high degree of complexity that results from the interplay of various CESs within a collaborative system group (CSG). CSGs are formed by CESs to achieve goals that individual CESs cannot achieve on their own. For example, collaborative autonomous transport robots (CESs) can form fleets (CSGs) to optimize the transportation of goods in a factory. In a CSG, it is not only the CESs that have goals; the CSG also has goals which in turn result from the goals of the CESs. However, the different goals may be partially contradictory. For example, an individual robot may be interested in conserving its battery life, while the fleet as a whole is interested in minimizing disruption in production. The decentralized organization and the dynamicity of such CSGs (for example, robots may join or leave the fleet during runtime) makes them highly complex and their development challenging.

Goal-based strategy exploration

Therefore, we propose a goal-based approach for strategy exploration that considers the peculiarities of CSGs and CESs. In detail, we introduce a goal modeling approach tailored to the specification of goals for CESs and CSGs and show how strategies can be developed based on these goals and operationalized. We demonstrate the applicability of our approach for a case example from the industry automation domain. Specifically, we illustrate the impact that different strategies for a fleet of autonomous transport robots have on the fulfilment of goals by the individual robots.

9.2 Goal Modeling for Collaborative System Groups

Goals document objectives and rationales

Goal models are used—often in the early development phases—to document objectives that a system under development should achieve [van Lamsweerde 2001]. Goals typically document the rationales for more concrete and technical requirements, as well as design decisions [van Lamsweerde 2001], [Yu 1997]. For a recent overview of goal modeling, please refer to [Horkoff et al. 2017]. While several goal modeling languages have been proposed, we focus on the use of the Goal-oriented Requirement Language (GRL), which is part of the ITU-standardized User Requirements Notation [International Telecommunication Union 2012] and a good fit for CESs and CSGs (cf. [Daun et al. 2019], [Brings et al. 2020]).

Goal-oriented Requirements Language

GRL is based on the i* framework [Yu 1997] but is less restrictive regarding the use of the notational elements (cf. [Horkoff et al. 2008]). GRL encompasses the use of actors to denote stakeholders that have some goals to be achieved by using a system under development — that is, they may depend on other actors to achieve their goals. Most notably, in agent-oriented software engineering, actors are also used to model technical systems [Bresciani et al. 2004], as is the case here.

Intentional elements

In GRL, the intentions of actors can be further specified by means of several types of intentional elements. Intentional elements are subdivided into (hard) goals, soft goals, tasks, resources, and beliefs, which are related via decompositions and contribution links. In the following, we illustrate how the use of goal modeling can foster the engineering of CSGs.

Local goals and global goals

No reasonable system is supposed to behave absolutely arbitrarily. Each system has goals that it has to achieve. A CSG is formed through the cooperation of the CESs, therefore the goals of the CSG must be considered as early as during the design phase. The following distinction between goals is quite natural: we refer to *local goals* as goals of a CES and *global goals* as goals of a CSG. Global goals are goals that each CES of the CSG aims to achieve, while local goals are goals that represent the interests of individual CESs.

CSG goals vs. CES goals

This allows us to separate goals of the CSG from goals of the CESs but also to denote relationships between both — for example, to identify where the CSG depends on the individual CESs in its goal fulfillment and vice versa. This is important information, as we will see later on that these dependencies drive design decisions and result in the definition of explicit strategies. Figure 9-1 shows an excerpt from the GRL goal model of an individual transport robot. The goal model emphasizes the transportation-related goals of the CESs. The goal model depicts the robots' responsibilities for route calculation, performing transport tasks, and bidding for transport tasks. For each robot, detailed sub-goals are derived that, for example, define how charging is to be handled and how robot breakdowns must be treated.

Hierarchically structuring goals

As we can see, the goal model specifies three important high-level goals (which are defined directly after the root node) regarding the safety of humans, conducting the transport, and the robustness of the robot. These goals are then refined until fine-grained tasks are reached, such as the tasks to determine obstacle positions and to communicate the obstacle's position. These goals identified for the individual robots are closely related to the overall goals of the CSG—the fleet of robots—as each individual robot depends on the fleet of

www.dbooks.org

robots and vice versa. We investigated this issue in [Brings et al. 2019], [Brings et al. 2020] in more detail.

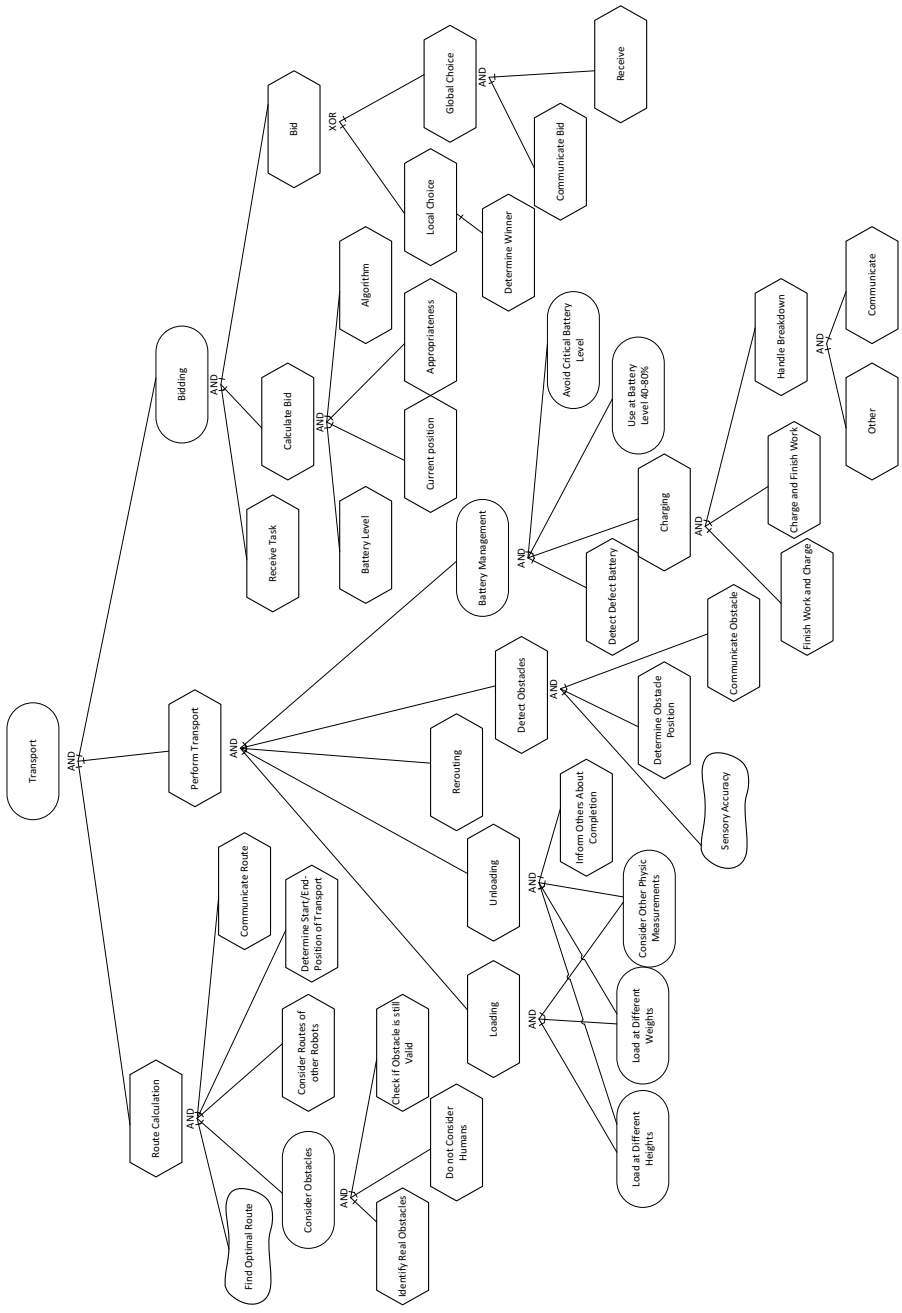


***Fig. 9-1**: GRL goal model for the individual transport robots*

9.3 Goal-Based Strategy Development

Collaboration enables goal achievement

Collaboration can enable embedded systems to achieve goals that cannot be achieved by a single system on its own. Having identified such goals (e.g., by using the approach from Section 9.2), one of the next engineering challenges is to develop behavior (or strategies) for the collaborating systems that will enable the goals to be achieved. In this section, we present concepts for going from identified goals towards behavior and we apply the concepts to an example in the use case of autonomous transport robots.

Starting with goals

The goals of a CSG and of the individual CESs describe what the systems should achieve at runtime. In general, those goals are described in a way that is understandable for humans. There is no specification of the conditions under which the goals are achieved or how to identify goal fulfillment. For example, for a goal such as "the maintenance costs of the system group are minimized," at design time, there is no information about which maintenance costs are minimal but still realistic: maintenance costs of zero are desirable but this cannot be achieved in a dynamic system. Additionally, there is no specified point in time for checking whether this goal is fulfilled and, furthermore, it is not clear what measurements are needed to predict maintenance costs.

Defining key performance indicators

Key performance indicators (KPIs) are used to make goal fulfillment measurable: KPIs relate goals to observable system variables and measure the degree to which goals are fulfilled over time. Each goal is reflected by at least one KPI in order to enable assessment of the system behavior at each point in time with respect to the goals. We look at KPIs in more detail in Section 9.4.

Defining strategies

We describe system behavior in terms of strategies: from a formal perspective, a strategy of a system is a function that maps the history of the system behavior and its context to a (new) valuation of the system variables. In other words, a strategy is an instruction for the system regarding how to behave in any situation it could face. When designing a collaborative embedded system, we aim for strategies that ensure the fulfillment of all goals "as well as possible," using KPIs to measure the degree of goal fulfillment.

Conflicts

Due to potential conflicts between goals and unpredictable context behavior, it is not always possible to find a strategy that fulfills all goals completely. Hence, we have to decide which strategy is the best match for the goals, even if there is no perfect solution. The definition of a quality measure for strategies supports this decision: a quality measure is a partial order relation on the set of all strategies for a

www.dbooks.org

system. In particular, a quality measure is a function that takes two strategies of a system and decides whether one of those strategies is better than the other one. Not all strategies are comparable, hence the quality measure is only a partial order. By deciding whether one strategy is better than another one, the quality measure resolves potential conflicts between goals — for example, by prioritizing the list of existing KPIs. The relationship between goals, KPIs, and quality measures is summarized in Figure 9-2.

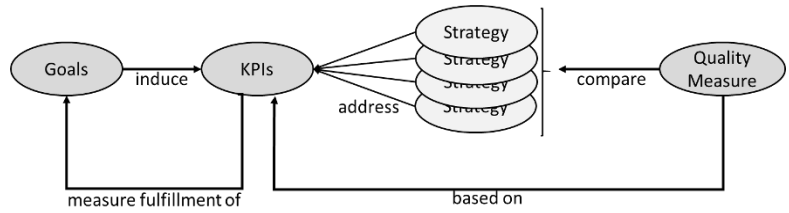


***Fig. 9-2**: Relationship between goals, KPIs, and quality measures of strategies*

In the following, we present a proof of concept example that illustrates the benefit of defining KPIs and quality measures based on goals for finding appropriate strategies.

Application to the autonomous transport robots

In the use case of autonomous transport robots, one of the key objectives is the transformation from a central fleet management to a decentral one. This requires a definition of how the robots distribute transport tasks among the fleet such that not only goals of individual robots, but also goals of the complete fleet, are fulfilled. In this example, we focus on the distribution of transport tasks and the global goal of having equal wear and tear among all robots of the fleet. This goal has industrial relevance, since it supports predictive maintenance and a reduction in maintenance costs for the robots — since all robots can be maintained in a single appointment with a service team instead of needing a service team every time a robot reaches some given threshold for distance driven.

Making goals measurable

To make this goal measurable, we define a KPI for the difference between distances driven per robot. More precisely, we observe the difference between the robot with the least distance driven and the robot with the most distance driven. To keep the example simple, we omit additional KPIs that may also be relevant for the goal. We also focus on the decision on the distribution of transport jobs. Here, therefore, a strategy for a transport robot is determined completely by defining which transport tasks the robot takes over.

Quality measures for strategies

We define the quality measure for strategies as follows:

A strategy s is better than strategy u if the KPI "difference between the robot with the least distance driven and the robot with the most distance driven" after fulfilling a task is smaller (or equal) when applying s than when applying u.

Alternative strategies

We define three alternative strategies for task distribution among robots:

- Strategy 1: The robot with the lowest distance covered so far takes over the job.
- Strategy 2: The robot with the lowest additional distance to cover for the task fulfillment takes over the job.
- Strategy 3: For each robot, calculate the difference in the distance covered if the robot takes over the job. The robot with the smallest calculated difference value takes over the job.

Comparing strategies

Considering the quality measure introduced above, we can use examples to show that the first two strategies are not comparable. Furthermore, we can formally show that the third strategy is better than the first and the second one. This qualitative comparison can be complemented by a quantitative simulation-based comparison: we used a MATLAB model of the fleet of robots and generated 100 random topologies for the factory, each defined by the distances between relevant locations in the factory, such as machines, charging stations, and storage facilities. Each factory generated consists of 5 to 30 relevant locations. The distances between locations were chosen at between 5 and 50 meters. For each of those factory maps, we generated a list of 100 transport tasks between locations of the respective factory. Each of the three strategies for distributing the tasks among the fleet of robots was applied. The number of robots per fleet was chosen randomly as between 2 and 20. As the initial state of the fleet, each robot was randomly set to one of the locations, and an initial value for the distance already covered was chosen randomly as between 0 and 200 meters. After each task distribution, the difference between the minimum and maximum costs of the robots after fulfilling all tasks they took over was calculated according to the quality measure. The simulation showed the following results:

1 In the simulation, we were able to verify that strategy 3 performs better than the two other strategies with regard to the quality measure defined above. Additionally, the

www.dbooks.org

simulation showed that strategy 1 and strategy 2 are not comparable.

2 As a quantitative result of the strategy comparison, we observed that the average differences between the maximum and minimum costs of robots of the same fleet is the smallest if the robots behave according to strategy 3: the mean cost difference between robots with strategy 3 is just 81% of the cost difference between robots with strategy 1, and just 88% of the cost difference between robots with strategy 2.

Figure 9-3 shows the evolution of the average difference between the highest and lowest costs of robots over the number of distributed transport tasks for the three strategies. The figure illustrates that strategy 3 performs the best with regard to minimization of cost differences. Strategy 1 has the least convincing performance.

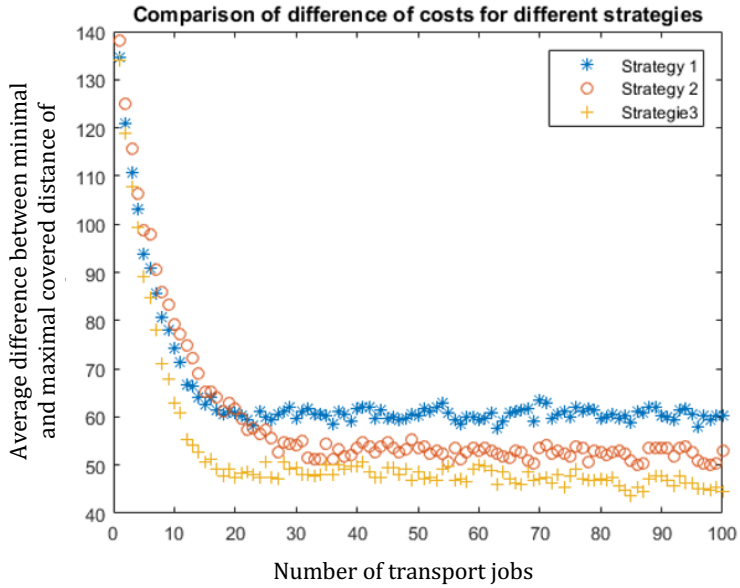


***Fig. 9-3**: Simulative comparison of cost differences for different strategies*

Choosing strategies impacts fleet performance

Our simulation shows that choosing the right strategy has a measurable impact on the performance of the fleet of robots. An explicit definition of a quality measure for strategies in the early design phase allows us to identify good strategies that had not been thought of before (here, Strategy 3, the best of the strategies considered, was introduced as a new strategy after the definition of the quality measure). Hence, considering strategies and quality

measures in the development of autonomous transport robots is a method that helps to improve the performance of the fleet significantly. In our evaluation, the fulfillment of the optimization goal "equal cost distribution among the fleet" has been improved by nearly 20%.

9.4 Goal Operationalization (KPI Development)

Developing a fleet of robots capable of performing transport jobs without central management necessitates good tracking of fleet/robot performance in fulfilling a set of goals.

Goals in the fleet of robots

The fleet of robots must fulfill multi-level goals. Some of the goals must be fulfilled absolutely, while others can be fulfilled in relation to other goals. This defines some kind of trade-off between these goals that have no conflicts with each other. In other words, the level of fulfillment of these goals must be measured and analyzed at runtime to evaluate the strategy performance that defines how and with which priority these goals must be accomplished.

Determining goal fulfillment using KPIs

In order to measure the fulfillment of the goals as well as determine how well the fleet/automated guided vehicle (AGV) is performing, we have to define KPIs. These KPIs serve as feedback data to the strategy components, which can lead to strategy adjustments to achieve better goal accomplishment if the present accomplishment is not good enough. In a multi-level goal system, it is helpful to categorize the KPIs. These categorizations are specific to a use case and would make it easier to define the trade-off between goals. In other words, this would give indications of the prioritization of such a goal, as well as of the definition of the interconnections between the goals and also the impact of not achieving them among each other. An example of KPI categorization for autonomous transport robots is as follows:

- Local/global KPI
- Historical/real-time KPI

Definition 9-4*: Local KPIs*

The term local KPI refers to the indicator that reveals how much a local goal is fulfilled during the performance of the robot.

Definition 9-5*: Global KPI*

The term global KPI refers to the indicator that reveals how much a global goal is fulfilled during the performance of the fleet of robots.

www.dbooks.org

Definition 9-6*: Historical KPI*

These KPIs reveal the quantity of goals fulfilled during a certain period of the fleet performance — for example, one week. These KPIs can be used for long-term analysis and adjustments of the strategy catalog at design time.

Definition 9-7*: Real-time KPI*

These KPIs reveal the quantity of goals fulfilled at runtime. These KPIs can be used in analysis to adjust a strategy at runtime.

Example 9-8: Decentralized fleet of robots

As an example, the following table shows a list of goals and their relevant KPIs.

Table		KPIs	
Goals	Battery health and safety	Minimum/maximum cell voltage [V]	Real-time/local
	Equal wear and tear	Distance covered by each robot in a given time frame compared to the distances covered by other robots	Real-time/local
	As soon as possible job fulfillment	The difference between the earliest pick-up time and real pick-up time [s]	Historical/global
	As soon as necessary fulfillment	The time frame between real delivery time and latest due date per transport job [s]	Historical/global

A set of goals for the autonomous transport robot use case is as follows:

- **Battery health and safety**: Transport robots must not let their battery deplete or overcharge.
- **Equal wear and tear**: The transport robots must operate in such a way that all transport robots of the same age show approximately the same usage.

- **As soon as possible fulfillment**: The fleet of robots must fulfill all incoming jobs as soon as they are requested.
- **As soon as necessary fulfillment**: The fleet of robots must fulfill all incoming jobs exactly at the time they are expected to.

By formalizing the KPIs identified and implementing them in a monitoring tool, we can keep track of how well a strategy is fulfilling the desired set of goals.

9.5 Modeling Methodology for Adaptive Systems with MATLAB/Simulink

Adaptive systems face a plethora of complex scenarios to be accounted for

The development of CESs/CSGs requires a well-founded approach for dealing with a number of difficulties that result from the high complexity of the scenarios involved and that have to be incorporated. For instance, an autonomous fleet of robots must react to dynamic changes in the policy of the manufacturing execution system (MES), or the number and nature of its members, in such a way that the overall functionality and efficiency of the CSG is safeguarded. To give an example, the virtual exploration of strategies to address different goals is essential to improve a system's efficiency, cf. Section 9.2. In this context, the consistent application of a model-based development process for CESs offers a variety of benefits, such as early and systematic validation of functional requirements that describe the CSG/CES behavior. Different engineering solutions can be based on suitable system model variants that are validated and compared in a fully or partly virtual context. Moreover, Simulink models can interface with typical robot middleware or communication frameworks, such as the Robot Operating System (ROS). For instance, Simulink models may define ROS nodes or generate standalone ROS nodes based on C++ for use in an ROS network.

Need for tailored tool chains

The model-based approach greatly benefits from tailored tool chains, which automate a large number of development activities, including requirements management, modeling and simulation, as well as integrated quality assurance. For instance, in the case of the fleet of robots, the monitoring of the distribution process for incoming tasks can be automatically included in the Simulink model. The virtual representation provides a sound foundation for developing, maintaining, and extending the actual system and its hardware/software/mechanical components efficiently.

Using Simulink

With regard to the model notation, the domain-independent language Simulink is suitable for describing the functional behavior of

www.dbooks.org

the CSG and the CESs as well as their context. In the case of a fleet of robots, the manufacturing execution system broadcasts different global goals dynamically to the fleet of robots. Typically, the global goals define a trade-off between the following competing objectives:

1. Economy: Minimize the total distance driven by all CESs — i.e., the transport robots.
2. Robustness: Keep the job queue lengths of each robot as short as possible.
3. Performance: Maximize the number of jobs executed per time unit.
4. Maintenance: Distribute the tasks such that all robots drive a similar distance.

As mentioned in the preceding paragraphs, KPIs are used to represent the goals in a measurable way. A suitable collaboration strategy for the collaborative robot fleet members must be designed corresponding to the given goals, cf. Sections 9.2 and 9.3. Therefore, the fundamental part of the modeling is dedicated to the distribution of the incoming transport jobs depending on the dynamically changing objectives. The collaborative fleet of robots consists of a finite number of robots that redundantly control and maintain the required data structures, such as job queues, distances driven, and their batteries' states of charge. Based on this data, a bidding process determines the collaborative robot fleet member with the lowest job execution cost. The global goals are encoded using a suitable bidding parameter vector. The context model, which represents the highest level in the hierarchy of system models, describes the interaction of the transport robot with its environment — for example, the manufacturing execution system. Furthermore, a suitable transport robot architecture that is capable of addressing adaptivity can be introduced based on a hierarchical decomposition. This approach yields a decomposition-type model that defines each transport robot's components and interfaces. Most notably, the collaborative AGV controller (CAC) hosts the logic for calculating the bidding values based on the current system state and goals. Correspondingly, each CAC model consists of the following:

- A reconfiguration unit, which is triggered whenever a new transport job is published or the collaborative robot fleet constituents are altered

- A processing unit for the transport robot goals — that is, bidding values for the autonomous task distribution are computed from the CAC data, as well as from the bidding parameters associated with the currently active transport robot goal and the member-specific local goals (e.g., maintaining a minimum battery level)
- A bidding unit that determines which robot receives the published task
- A unit that holds and updates the CAC data (battery level, path lengths, etc.)
- Units that manage the interface with ROS to determine path lengths and battery states

Figure 9-9 shows the resulting components in the system decomposition model. The system behavior is fully composed from the behavior models of each component. These component-related behavioral models represent the third level in the hierarchy of system models.

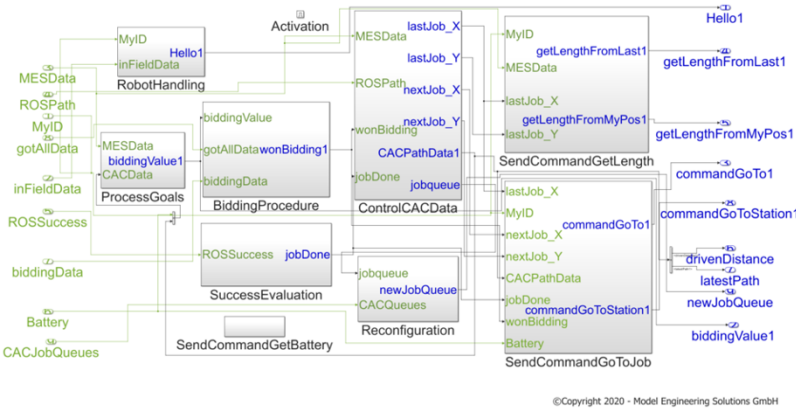


***Fig. 9-9**: System decomposition model in Simulink*

Capturing MES policies in the requirements

The expected adaptive system response, which is subject to dynamically varying manufacturing execution system policies, must be fully captured in the requirements of the fleet of robots. Compared to natural language-based approaches, which are still widely used in practice, formalized requirement formats give rise to unambiguous representations of requirements of the fleet of robots. Moreover, with the model-based approach, formalized requirement formats can be fully integrated in the sense that state-based or event-based triggers and the required signal response can be fully defined using references to model entities, such as signal specifications or design parameters.

www.dbooks.org

In conjunction with the efficient definition of appropriate test cases, virtual validation of adaptive CSG behavior can be automated based on automatic test execution and assessment. The assessment relies on the comparison of the logged output signals of the executable Simulink CSG model with the expected output signals as defined in the formalized requirement.

9.6 Collaboration Framework for Goal-Based Strategies

9.6.1 Fleet Management in Collaborative Resource Networks

Fleet management systems coordinate resource usage

A fleet management system of the transport robots coordinates and monitors the use and status of a CSG, including the offered functionalities emerging from the available resources. For example, a group of transport robots offering the operational resource of transporting items. In a collaborative, goal-based approach, these functionalities should be realized in a decentralized fashion and distributed to the transport robots so that they can be executed collectively in a fleet of robots. As mentioned before, this requires the ability of each CES to achieve its individual goals and to contribute in an optimal way to the goals of the fleet/CSG.

The collaboration framework provides generic functionalities

A collaboration framework provides the generic collaboration functionalities needed during development and operation of the CESs and the CSG. These functionalities support the CESs in making informed decisions. Each CES, thereby, decides independently and takes appropriate actions. This allows for self-governing and self-organizing functionalities to have secure and trusted interactions between the CESs in the CSG. Most importantly, the framework must provide the capability to set up and execute interactions and communication between the CESs in the CSG.

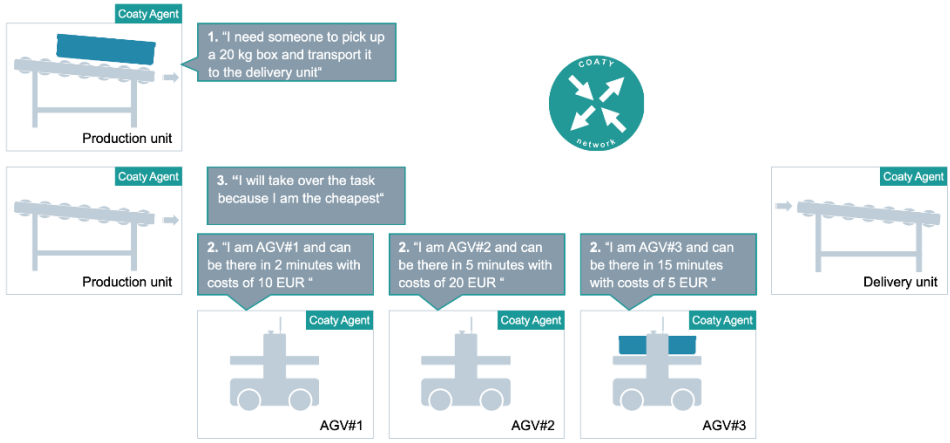


***Fig. 9-10**: Example AGV scenario for goal-based, collaborative fleet management*

Figure 9-10 shows an exemplary scenario for a collaborative, goal-based fleet management. The exemplary scenario represents a factory floor as the scope of a fleet of robots with two production units as transport robots, three AGVs as CESs, and one delivery unit as a CES. Products output by the production units must be transported to the delivery unit by autonomous transport robots. For this exemplary scenario and considering only the transport resource allocation, a collaboration framework must enable:

- ❑ The production units to announce new transport requests
- ❑ The robots to receive transport request announcements
- ❑ The robots and the productions units to coordinate the assignment and fulfillment of tasks
- ❑ All system components to monitor relevant information and behavior in order to elaborate on the level of achievement of individual and fleet goals

The following section explains how such a collaboration framework can be designed based on a set of communication patterns and a typed object model for a decentrally organized CSG. The communication patterns provide models of how the CESs can communicate and hereby interact with each other. The typed object model ensures that the basic communication data model between the CESs matches and is extensible.

www.dbooks.org

9.6.2 Collaboration Framework

A collaboration framework, like the one provided by Coaty (https://coaty.io), is designed to enable autonomous Internet of Things (IoT) devices, as well as people and services, to interact in changing scenarios. Here, we apply it to a self-organizing fleet management, whereby the following properties must be fulfilled:

- Loose coupling: CESs must be able to interact independently of each other. Thus, a tight coupling with other CESs or system components would hinder the collaboration approach. A preferable approach would not apply device-centric communication concepts but instead, a decentralized and data-centric concept based on an event architecture, as typically applied in publish-subscribe communication principles. This also allows CESs to participate in or leave CSGs on demand.
- Any-to-any communication: CESs must be able to interact in one-to-one, one-to-many, many-to-one, or many-to-many CES-to-CES communication scenarios. In addition, all communication scenarios should be available as one-way communication for publishing and subscribing information topics and two-way communication for request-response communication.
- Interoperability: Besides having a standard set of communication patterns for modeling the interaction between CESs, for interoperability, an extensible data model for CES interaction must be established.
- Collaboration functions: CESs must be able to take advantage of generic collaboration functions, such as negotiation or consensus finding. This is especially important to enable implementation of the multi-level goal strategies for CES and CSG.
- Programmability, extensibility, and portability: The collaboration framework must be designed in such a way that it can be easily extended and programmed.

The IoT framework Coaty

The exemplary collaborative IoT framework Coaty fulfills these key properties. It is based on a lightweight and modular architecture that allows extensibility by means of specific connectors, adapters, building blocks, etc.

Event-based communication flows

The framework uses event-based communication flows with one-way/two-way and one-to-many/many-to-many communication patterns to realize decentralized prosumer scenarios for CESs. It thereby combines the characteristics of both classic request-response and publish-subscribe communication, but maintains the data-centric and loose-coupling characteristics. In contrast to classic client-server

systems, all participants in the system are equal in that they can act both as producers/requesters and consumers/responders. These communication patterns (cf. Figure 9-11) allow data to be discovered, queried, shared, and updated on demand in a distributed, decentralized CSG. In addition, the collaborative IoT framework Coaty allows for a distributed implementation of triggering context-specific remote operations and dynamic context-specific information routing between CESs by its IORouting concept. The IORouting concept (https://coatyio.github.io/coaty-js/man/developer-guide/#io-routing) introduces a way to dynamically route information flows between information sources of a CES and information actors that use the information. This information routing takes place based on changes in the observed operation context of the CSG. The challenging issue of reaching programmability in such highly complex, distributed, asynchronous systems of CESs in a CSG is achieved by applying the reactive programming paradigm. The extensible typed object model applied, with a set of basic core object types, can represent domain-specific system characteristics such as tasks, etc. Furthermore, each CES is represented as an object such that interoperability can be maintained without losing extensibility.

Applying this kind of collaboration framework, a set of CESs that form a CSG can collaborate by means of a decentralized interaction and communication network.

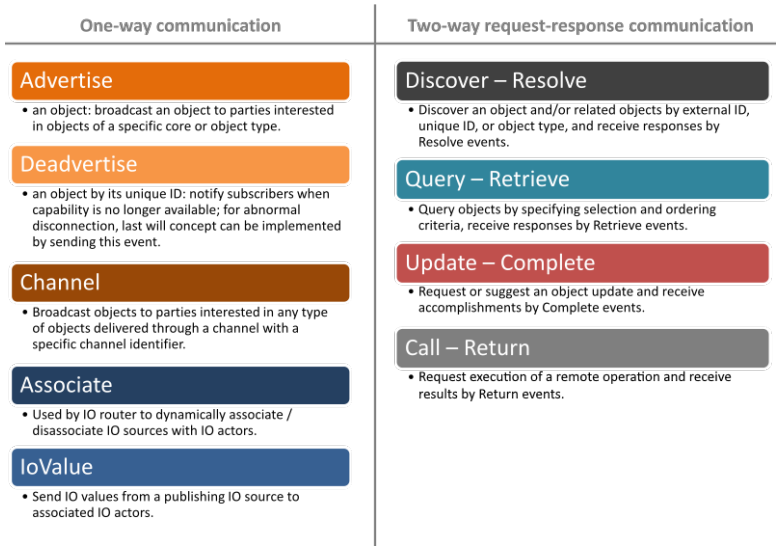


***Fig. 9-11**: Collaborative IoT framework communication pattern as realized in Coaty*

www.dbooks.org

9.6.3 Collaboration Design in Decentralized Fleet Management

The collaboration framework referred to above allows us to perform and model the collaboration design of a decentralized fleet management. The following five different functional areas must be designed for the collaboration:

1. Modeling and announcement of tasks to the fleet of robots with their functional and non-functional requirements to be executed
2. Observation of these tasks by the individual transport robots
3. Monitoring local system's and fleet of robots' states at each individual transport robot
4. Application of the transport robots' goals and the goals of a fleet of robots to calculate an offer for tasks
5. Decentralized coordination of the decision, based on the CES offers, about which CES receives the task

Let us consider the exemplary scenario from Figure 9-10; this could be designed in a simplified way as follows: all transport robots observe transport tasks and other relevant system states. All systems observe transport task bids. The production unit issues a new transport task, with weight, pick-up and drop-off positions, a bidding strategy, and a bidding period. The robots calculate a cost function based on their individual goals and the fleet goals and issue the result as a transport task bid to the CSG. Each robot evaluates the bids received for the defined bidding period and then decides whether it wins the negotiation. If it does, the CES announces the self-assignment of the task and the CSG places the task in its local job queue and executes the task in accordance with its priorities in the job queue.

As mentioned before, this scenario is very simplified and does not include any failure handling etc. It shows that transport robots can interact with each other in a fleet of robots using a collaboration framework in a powerful way, allowing collaborative goal-based strategies to be modeled and implemented in a structured way that can be validated.

9.7 Conclusion

CESs connect to form a group in order to achieve local and global goals by following the best possible strategy. The interconnection between goals, KPIs, monitoring, and strategy shapes the core concept of the

goal-based strategy exploration method. In this chapter, we presented the concepts for moving from identified goals towards strategy development. We then applied the concepts to an example from the use case of collaborative autonomous transport robots. In doing so, we focused on the challenge of how to develop a set of strategies in which multi-level goals must be achieved. Therefore, the goal fulfillment must be measured and qualified for each strategy.

We also introduced the modeling tool and collaboration framework to support the application of this approach to an industrial use case to reveal some of the challenges in forming such a CSG.

9.8 Literature

[Bresciani et al. 2004] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos: Tropos: An Agent-Oriented Software Development Methodology. In: Autonomous Agents and Multi-Agent Systems 8 (3), 2004, pp. 203–236.

[Brings et al. 2019] J. Brings, M. Daun, T. Bandyszak, V. Stricker, T. Weyer, E. Mirzaei, M. Neumann, J. S. Zernickel: Model-Based Documentation of Dynamicity Constraints for Collaborative Cyber-Physical System Architectures: Findings from an Industrial Case Study. In: Journal of Systems Architecture - Embedded Systems Design 97, 2019, pp. 153–167.

[Brings et al. 2020] J. Brings, M. Daun, T. Weyer, K. Pohl: Goal-Based Configuration Analysis for Networks of Collaborative Cyber-Physical Systems. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, SAC '20. Brno, Czech Republic, 2020, pp. 1387–1396.

[Daun et al. 2019] M. Daun, V. Stenkova, L. Krajinski, J. Brings, T. Bandyszak, T. Weyer: Goal Modeling for Collaborative Groups of Cyber-Physical Systems with GRL: Reflections on Applicability and Limitations Based on Two Studies Conducted in Industry. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019.

[Horkoff et al. 2008] J. Horkoff, G. Elahi, S. Abdulhadi, E. Yu: Reflective Analysis of the Syntax and Semantics of the I* Framework. In: Advances in Conceptual Modeling – Challenges and Opportunities. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2008, pp. 249–260.

[Horkoff et al. 2017] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, P. Giorgini: Goal-Oriented Requirements Engineering: An Extended Systematic Mapping Study. In: Requirements Engineering, September 2017, pp. 1–28.

[International Telecommunication Union 2012] International Telecommunication Union: User Requirements Notation (URN), Z 151, 2012.

www.dbooks.org

[van Lamsweerde 2001] A. van Lamsweerde: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, 2001, pp. 249–62.

[Yu 1997] E. S. K. Yu: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997, pp. 226–235.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Samira Akili, Humboldt Universität zu Berlin
Emilia Cioroaica, Fraunhofer IESE
Thomas Kuhn, Fraunhofer IESE
Holger Schlingloff, Fraunhofer FOKUS

10

Creating Trust in Collaborative Embedded Systems

Effective collaboration of embedded systems relies strongly on the assumption that all components of the system and the system itself operate as expected. A level of trust is established based on that assumption. To verify and validate these assumptions, we propose a systematic procedure that starts at the design phase and spans the runtime of the systems. At design time, we propose system evaluation in pure virtual environments, allowing multiple system behaviors to be executed in a variety of scenarios. At runtime, we suggest performing predictive simulation to get insights into the system's decision-making process. This enables trust to be created in the system part of a cooperation. When cooperation is performed in open, uncertain environments, the negotiation protocols between collaborative systems must be monitored at runtime. By engaging in various negotiation protocols, the participants assign roles, schedule tasks, and combine their world views to allow more resilient perception and planning. In this chapter, we describe two complementary monitoring approaches to address the decentralized nature of collaborative embedded systems.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_10

www.dbooks.org

10.1 Introduction

In its most general meaning, *trust* is the belief of one agent in the capabilities and future actions of another agent. Relying on this belief, the trustor hands over control to the trustee and faces negative consequences if the trustee does not perform as expected. In collaborative embedded systems (CESs), trust is important on several levels, as depicted in Figure 10-1. Firstly, the components of the collaborative system group (CSG) need to trust each other in order to pursue common goals. Secondly, in safety-critical contexts, the (human) user needs to trust the CSG to work as specified, and the CSG itself needs to trust its environment to behave as laid down in the specification. Thirdly, as each CES in the group may consist of components from many different vendors, it needs some self-reliance, that is, trust in its own components.

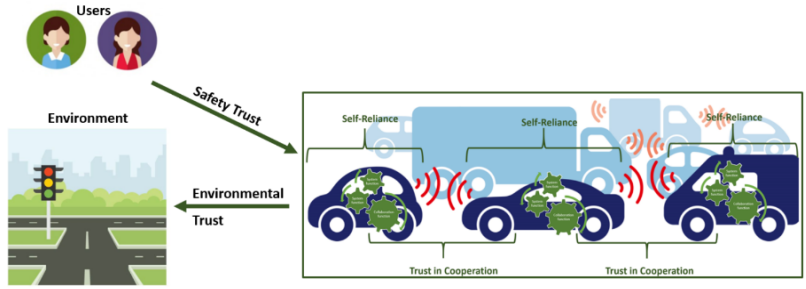


***Fig. 10-1**: Aspects of trust around CESs*

Besides the question "Who trusts whom?", the question "Why trust?" defines another dimension in the analysis of trust. Trustworthiness can be established by a trustee in several ways: via certificates from trusted third parties, via a history of reliable actions, or by giving insights into its decision-making process. In the following, we comment on each of these in the context of collaborative embedded systems. *Certificates from trusted third parties* are used to increase the trustworthiness of the trustee via the reputation of the certifying institution. For example, an autonomous car would not be allowed to enter a platoon if the software has not been certified by the respective authorities. Certificates are usually issued for the design of a system. At runtime, if certificates are used, there must be a mechanism that can show that the certificates are original and unmodified.

A *history of reliable actions* can be established at design time — for example, by means of extensive testing. This is the preferred way if the system is deterministic, that is, in any given situation, it has a unique, reproducible behavior. However, when nondeterministic agents have to negotiate in their operation, this history is primarily established during runtime. For example, in a group of transport robots bidding for a certain job, a robot may be singled out if it has a "bad reputation" of not accomplishing jobs on time. In game theory, several scenarios, such as "tit for tat" and the "prisoner's dilemma," have been investigated to develop a theory of trust in the presence of competition.

Insights into the decision-making process is a trust-building measure because it allows the trustor to predict the actions of the trustee in advance. For collaborative embedded systems, this can be realized by having each agent communicate not only decisions and actions, but also goals, plans, and other reasons. Since the decision-making process takes place at runtime, this communication is inherently dynamic.

In the rest of this chapter, we elaborate on three methods for building trust in collaborative systems. In Section 10.2, we describe an architectural pattern that can be used for the certification of systems at design time. In Section 10.3, we describe a method of predictive simulation that allows trust to be built at runtime. In Section 10.4, we describe online monitoring as a method for extrapolating future behavior of a system from its past and present actions.

10.2 Building Trust during Design Time

In this section, we introduce the concept of a prototypical platform that supports certification of software behavior. Trust at design time is then built by verifying software execution in a multitude of scenarios.

The introduction of autonomy into technical systems brings new challenges for safety and security. Since the majority of accidents on the roads are caused by driver error, one way of increasing safety is to take away some of the driver's responsibilities. However, such autonomy is only permissible if a corresponding trust can be established in the technical components. If the level of autonomy is increased by the integration of third-party components, additional trust checks are required. This is necessary because a software component delivered as a black box can contain logic bombs

www.dbooks.org

[Avizienis et al. 2004]. A vehicle that is part of a platoon is a collaborative embedded system designed to be under the control of a collaboration function. This collaboration function can negotiate tactical goals with other vehicles, such as the creation of a vehicle platoon. After an agreement on common goals has been reached, system functions that follow the agreed goals are activated.

However, even though a collaborative system's interaction with other systems happens at runtime, its safety architecture is decided in early development stages, at design time. A testing environment must therefore provide the ability to evaluate the system behavior in interaction with other systems whose behavior is unpredictable. Having a high number of successful test scenarios gives a high confidence that the CES will behave as specified during runtime and therefore deserves trust. For example, in the automotive domain, the behavior of a vehicle in a platoon must be tested in a high number of scenarios with other cars in order to give confidence that it complies with functional and non-functional specifications for platooning. Testing billions of scenarios on the road with actual cars is not feasible. Therefore, testing the system's behavior in simulated scenarios is imperative.

Design time verification requires testing in an extended set of scenarios

The testing framework we present in this section allows a high number of test scenarios to be executed for collaboration functions. Evaluation is performed in a virtual environment using simulation. The modular architecture of the framework allows the evaluation of additional software components of other autonomous systems, such as robots.

In the area of testing collaborative systems, existing approaches propose evaluation of the architecture of the ecosystems formed around them. In addition to the systems and components involved in an operational collaboration, the ecosystem contains actors that make the technical collaboration possible and also benefit from it, such as organizations, users, and developers. In these approaches, the evaluation is done by measuring the health of these ecosystems [da Silva Amorim et al. 2017], [da Silva Amorim et al. 2016]. The main aspects for evaluating the health are robustness, productivity, and niche creation. In contrast to these approaches, we evaluate collaborative systems by considering the quality of service. When systems start to collaborate, the collaborative group presents a new interface to its environment. With a visualization tool, we provide easily understandable information about the effects of interactions between systems. The information demonstrates the effects of

emergent services that can influence the health of the whole ecosystem.

In [Kephart and Chess 2003], autonomous elements mutually provide and utilize services in order to achieve individual goals. The vision is to have flexible relationships between autonomous software agents, with these relationships being established via negotiations. Relationships are represented by service provisions, and an independent manager oversees the agreements. This approach is oriented towards analysis of agents' interaction in an ecosystem. It provides a good base for reasoning about a system's interactions. The approach we present complements this work by providing a testing framework for analyzing the effects of collaborations.

Testing framework for CSGs

The testing framework follows the model view controller [Krasner and Pope 1988] architectural pattern, which is explained below. This allows modular components that can be exchanged when technical advancements are made. It also supports the reuse of components. The framework supports testing of collaborative embedded systems in holistic scenarios. These scenarios are formed with the help of digital twins. A digital twin is a simulation model of some embedded system in the real world that is linked to this system throughout its lifetime. The digital twins accurately represent the effects of actions and predicted intentions of a collaborative embedded system (CES) in the collaborative system group (CSG). The framework displays the effects of decisions taken by collaboration functions. In our context, a digital twin comprises real-world data and simulation models. The simulation models accurately represent the physical process of a real-world device. For example, within a platoon, the lead vehicle decides to increase the speed. The task of the collaboration function of a follower vehicle is to adjust the speed accordingly. In our testing framework, the lead vehicle and other cars are pure virtual entities for testing this collaboration function. For the follower vehicle, we have an actual ANKI car [ANKI 2020] (a model car on a scale of 1:10, with on-board electronics) that provides real-time data such as speed and position. We create a digital twin of this vehicle by combining a coarse simulation model with this data. In the framework, the behavior of the collaboration function can be observed via the digital twin. In contrast to a purely virtual approach, our framework allows the interaction between the hardware and the software to be tested in the physical car.

www.dbooks.org

Model

Model view controller [Krasner and Pope 1988] is an architectural pattern that divides the function of a framework into three components. We will demonstrate how this pattern can be used for testing CSGs. The functionality of a testing framework is to allow creation or integration of simulation models of CESs, definition of scenarios, execution of test cases, and evaluation of results.

The basic task of the modeler component is to provide an editor for the definition of pure virtual entities of the CSG. Moreover, a digital twin—that is, the combination of real-world data with a coarse behavioral model of a CES—can be created in this component. This modular structure allows simple and interchangeable units. Both pure virtual entities and digital twins can be represented as functional mock-up units (FMU) that can be executed in combination by a co-simulation platform.

Combining the real world with the virtual world

As a concrete implementation of this concept, Fraunhofer FERAL [Kuhn et al. 2013] is a simulation environment used for rapid development of architecture prototypes through coupling of simulators, simulation models, and high-level models. It enables abstract simulation models to be coupled with very detailed simulation models and digital twins. The integration of virtual agents and digital twins allows the evaluation of controlled decisions of real cars in an extended set of scenarios. The simulator provides the necessary environment for simulating and running the behavior of multiple virtual CESs.

As an example of a real-world agent, ANKI cars are small-scale model vehicles that can be programmed using the ANKI Software Development Kit (SDK). This SDK provides access to the sensors and actuators, and also to some higher-level functionality of the ANKI cars. Each ANKI car is equipped with infrared sensors that read encodings embedded in the track. Figure 10-2 shows the underside of an ANKI car. The infrared sensor is positioned at the front and the drive motor at the rear.



***Fig. 10-2**: ANKI car/real-world agent*

An additional Bluetooth Low Energy (BLE) module enables a duplex connection between every physical ANKI car and the SDK running on a Linux machine. Messages through the BLE connection go in two directions: commands from the simulator are sent from the simulator to the ANKI car via the SDK, and position information is sent back to the simulator. Position data consists of a combination of lane and segment numbers, with this data being obtained by the infrared sensor whenever the car crosses a checkpoint on the track.

View

The visualization engine of our framework receives information from the modeler component. It displays the results of a co-simulation by animating objects that reflect the dynamics within a test scenario. Since modularization is at the component level in our approach, the interfaces are complex. For an accurate representation of the behavior of the CSG, a high amount of complex information is necessary. The co-simulation platform produces information about the behavior of the CSG with a variable degree of accuracy that can be adjusted according to the testing intentions. If, for example, visualization of the effect of a communication failure in a platoon is intended, then messages describing this failure must be produced in the co-simulation framework. In the visualization engine, the failure can be displayed via a red alert symbol, for example. This means that it is possible to "zoom" into specific details of the simulated scenario. However, this possibility is limited by the bandwidth and computation power available.

As a concrete implementation of the view component of a testing framework, the Unity 3D game engine can provide a meaningful visualization for the scenarios and decision effects. For example, if a control decision has the effect of leading to a crash, this will be shown in the simulation. The modeler and view component can be combined with the observer design pattern. This is a behavioral pattern in which a *subject* maintains a list of *observers* and notifies them of any state changes by calling one of their methods. In our context, the subject is the message sent from the modeler to the view component. Each CES is an observer that reacts to this message by updating its state (i.e., position, speed, and acceleration).

www.dbooks.org



***Fig. 10-3**: Evaluation scenario visualized in Unity 3D from both a bird's-eye view and first-person perspective*

Controller

In our framework, the controller is the component that interacts directly with the user via web services. Through the controller, the user can define scenarios for the evaluation. The controller sends information about these scenarios to the modeler. It provides a service to the real-world object, which contains information about the pure virtual objects in the CSG. Other services include simulated sensor and actuator values. These services can be combined through service compositions. For example: the CACC (collaborative adaptive cruise controller) in a car can subscribe to a service giving GPS coordinates and to a service for the rotational speed of the wheels, and can thus provide a service of reference acceleration. These services are defined and composed in the controller and then passed to the modeler.

As an implementation example, Google Blockly [Blockly 2020] provides an intuitive framework for the definition of test scenarios. It provides a language of blocks, where each block represents a possible step in a test case. The semantics of a block can be defined in a suitable programming language. The test designer can use drag and drop to form complex test cases from the blocks. In our testing framework, this graphical modelling of a test case is transformed into JavaScript code that is parsed by our co-simulation tool FERAL. From there, it is

used to drive the Unity3D visualization. Figure 10-4 presents part of a test case that describes the behavior of two virtual cars in a platoon.

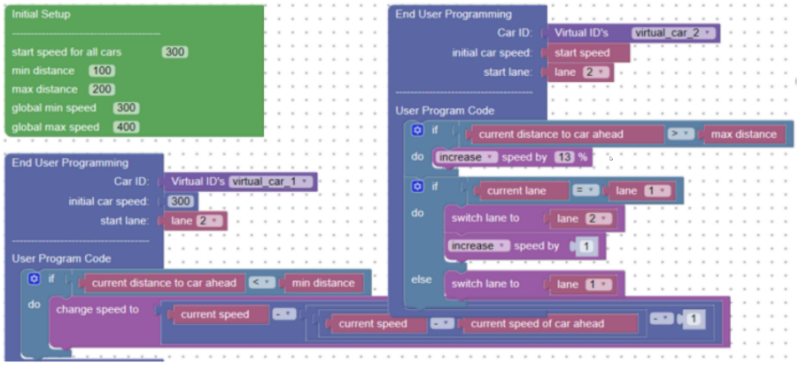


***Fig. 10-4**: Control algorithm of one virtual car*

In this section, we have shown how to combine real-world and virtual-world entities in order to test a CSG. The collaborative behavior of one CES in the group is tested in the simulation, whereas its physical behavior is tested on the actual hardware platform. This allows us to explore a wealth of collaboration scenarios with real-world components without the risk of damage to the actual hardware.

10.3 Building Trust during Runtime

The previous section exhibited an approach and a prototypical implementation for building trust at design time. However, some aspects of trust can only be built during runtime, since not all operational context can be foreseen in the design. In this section, we describe a method of predictive simulation that allows trust to be built at runtime.

During runtime, trust can be built through the addition of *predictive simulation* and *dynamic safeguarding* on the CESs. For this purpose, a software component simulating some aspects of the behavior of a CES is used. The abstraction can be with respect to three different aspects: timing behavior, functional behavior, and communication behavior. In order to allow an efficient online evaluation, only parts of the behavior should be modeled. With suitable abstraction, the simulation can be executed faster than the actual system behavior. It is therefore possible to foresee some effects of decisions before they are implemented in the real world. Moreover, the behavior of the simulated objects can be compared with the actual

www.dbooks.org

behavior of the physical entities. We can therefore detect hardware issues before they lead to problems. Such an approach requires the evaluation of the collaboration behavior at runtime. Predictive simulation and dynamic safeguarding can be used to build trust between the collaborative systems. For example, in a platoon, the follower vehicle needs to trust the lead vehicle not to make an emergency brake without a previous alert. Both the lead vehicle and the follower vehicle can run a simulation of the collaboration function. The follower vehicle can use a predictive simulation to calculate expected behaviors of the lead vehicle; if the lead vehicle behaves as expected, this increases its reputation. Therefore, the other vehicles may, for example, decrease the safety distance in the platoon. The lead vehicle itself can use dynamic safeguarding of its behavior. For example, it can simulate the collaboration function with respect to emergency braking and alerting. If it detects that there might be an emergency brake without prior alert, it can trigger an operational failover procedure that, for example, sends an alarm to the other cars. With this kind of runtime monitoring, it can increase its overall trustworthiness.

Predictive simulation is applicable for collaborative embedded systems in various domains. In the following, we focus on the specific context of automotive software engineering. In order to build trust, we can evaluate the collaboration function of a connected vehicle in a runtime predictive simulation. The collaboration function is deployed on the vehicle together with its corresponding abstractions. Complementary to the original algorithm, an abstraction defines an acceptable behavior range of output values for each combination of input values and internal state of the algorithm. When the car is driving on the road, the abstract behavior is continuously evaluated in simulated scenarios, where the simulated environment is an abstraction of the actual environment as observed by the sensors of the car. Correctness and trustworthiness of the collaboration function are validated by observing the effects of the simulation. In our work, we consider a distinction between correctness and trustworthiness. A software component that successfully passes all systematic tests and shows a correct behavior may still not be worthy of trust. This can happen if, at a later point in time, the software component shows an unexpected malicious behavior because of hidden timing bombs [Avizienis et al. 2004]. This means that the behavior is evaluated in a secured virtual environment (Figure 10-5, phase 1). Since the simulation is faster than the real evolution of the scenario, possible errors in the implementation of the collaboration function can be

detected in advance and protective measures can be taken. For example, if a car in a platoon receives an alert from the lead vehicle while leaving the platoon, the simulation could show the effects of neglecting the alert.

Dynamic safeguarding builds trust in the conformity of the collaboration function with its abstract representation (Figure 10-5, phase 2). This technology requires the parallel execution of the collaboration function and its abstractions (timing behavior, functional behavior, and communication behavior). Conformity is checked by comparing the actual behavior of the software with the ranges allowed by the abstraction. For example, if there is an emergency braking in the platoon, each car must apply a very accurate force to the brakes in order to avoid a collision with the preceding or succeeding car. The simulation could check whether the actual force applied to the brakes is within the force limits that were previously validated.

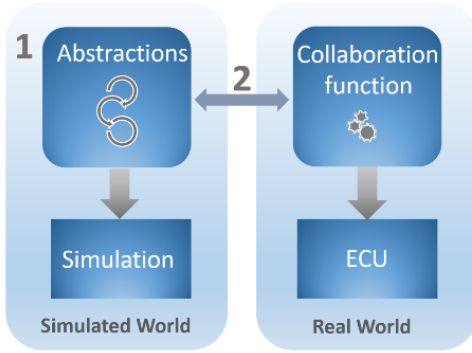


***Fig. 10-5**: Phases of the runtime trust evaluation method*

Predictive simulation can be realized with two possible strategies. Firstly, it can be based on a set of well-defined situations that evaluate the behavior in a virtual environment. Secondly, linked predictive simulation virtualizes the vehicle's current situation and predicts sensor data to reflect a forecast situation from the near future. Linked predictive simulation evaluates the abstractions in situations that are not covered by the first strategy. For example, in a platoon, when the lead car approaches an obstacle, we can monitor the abstraction of the collaboration function that sends adjusted desired speed commands to the following vehicles. If we observe that the collaboration function fails with this task, there is a big problem. Usually, today, this is solved by handing control back to the driver. Therefore, the lead car needs sufficient time to possibly override the decisions of the collaboration

www.dbooks.org

function if they are detected to be faulty. Thus, the execution of predictive simulation must be fast enough to allow operational failover solutions.

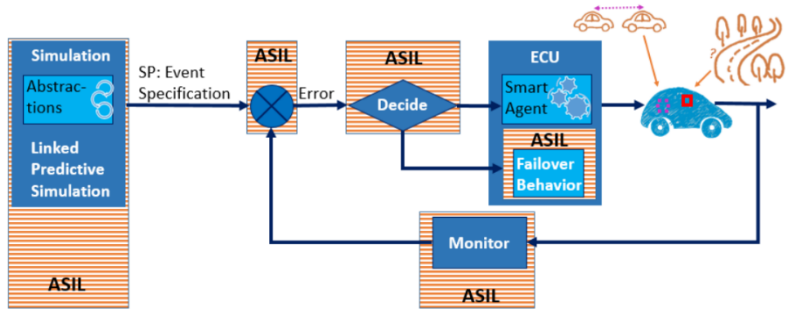


Fig. 10- 6: *Platform concept*

Figure 10-6 depicts predictive simulation and dynamic safeguarding in a closed control loop. The abstractions of the collaboration functions are executed in a secured simulated environment. During this predictive simulation, the order, type, and number of events are recorded and form the reference to which the actual execution of the software function on the electronic control unit is compared. The deviations between the expected behavior and the actual behavior are fed to a decision component that decides who controls the vehicle. If considerable deviations are detected, the execution of the software function is stopped and a higher trusted failover behavior is executed instead.

The software function is the subject of trust evaluation. Implementation of the method on safety-critical systems requires trusted design and verification of the platform components with appropriate ASIL (automotive safety integrity levels) set for each of them. Predictive simulation and dynamic safeguarding are a means to increase the trust and safety of the collaboration in a CSG. At the core of these methods is an abstract function description that is monitored during runtime. In the following, we elaborate on approaches that deal with monitoring the actual system behavior with respect to a formal specification.

10.4 Monitoring Collaborative Embedded Systems

While the above approach requires a full-scale system model in order to be able to override faulty system behavior, this may not always be

feasible. In this section, we present runtime verification as a lightweight method of monitoring a system for correct and safe operation. The general assumption is that a human supervisor can intervene and start a recovery routine if some faulty runtime behavior is detected. The runtime verification methods we present can be used to establish trust of a user in the CSG. As in the approach above, this is achieved by giving insights into the decision-making process.

There are manifold sources of runtime faults of an embedded system, and even more of a collaborative embedded system group (CSG). Within such a system, we have to deal with problems stemming from coordination and communication, concurrency, conflicting goals, and more.

In the remainder of this chapter, we describe the basic concepts of runtime monitoring and identify the challenges of applying it to collaborative embedded system groups. We then introduce two techniques that address some of the challenges identified.

Runtime Monitoring

Runtime monitoring is a popular approach for verifying the behavior of complex systems at runtime by checking the observed execution against a specification [Leucker and Schallhart 2009], [Bartocci et al. 2018]. This approach enables a fallback policy to be invoked if a deviation of the actual behavior from the specified behavior is detected. In the typical setup, the system under monitoring (SUM) is instrumented such that it emits signals or events that are processed by a monitor. The monitor, usually being much smaller and simpler to verify than the SUM, provides a formal guarantee of the detection of certain property violations. There have been many suggestions for specification languages, which vary in their complexity and expressiveness.

In general, there are two different approaches to constructing a runtime monitor for distributed systems. The monitor can be an additional computational entity of the system or it can be part of each component in the system. A centralized approach is often easier to implement, especially for systems already deployed. Furthermore, a centralized approach adds almost no computational overhead to each component. In contrast, a distributed approach scales naturally with an increasing number of components. This holds even if components are added dynamically at runtime. Moreover, there are applications (such as autonomous vehicle platooning) that are simply unfit for a monitoring third party.

www.dbooks.org

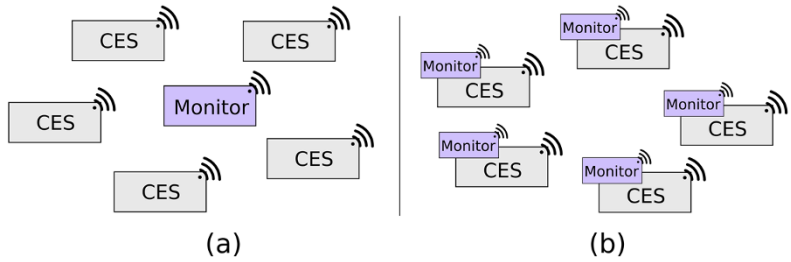


***Fig. 10-7**: (a) Centralized runtime monitoring (b) Distributed runtime monitoring*

Within the context of collaborative embedded systems, we are especially concerned with *distributed* runtime monitoring approaches. Since each CES in a CSG has its own goals and plans, it is more natural for a CES to also have its own monitor. Hence, in our approach, each component of the system is equipped with a monitor such that the monitors themselves build a collaborative system group (cf. Figure 10-7). In order to evaluate properties that rely on information produced by more than one component, monitors communicate by exchanging messages. Furthermore, a centralized monitor has to scale with the increasing number of systems at runtime and must be updated whenever a system with new capabilities (and thus new specifications) joins the collaborative group at runtime.

Runtime Monitoring of Collaborative System Groups

In a collaborative system group, collaborative embedded systems work together to achieve a shared goal and thereby provide a specific functionality. The successful completion of this core function requires collaboration, which is implemented by the use of interaction protocols for coordination or negotiation. As interaction protocols are thus the foundation of a CSG's behavior, the runtime monitoring of those protocols is at the core of our approach. Before providing an example and introducing two specification formalisms, we derive requirements for the runtime monitoring of CSGs:

Distributedness: To enable collaboration, CSG members exchange information via messages and perform local computations. If no global clock exists, asynchronous communication must be supported by the CSG architecture. Additionally, observable behavior can be described at the group level and at the individual level. While properties relating to the behavior of a single CES can be checked locally by monitoring methods for the verification of cyber-physical systems [Luckcuck et

al. 2019], the specification of the group behavior requires a language suitable for the expression of distributed system properties.

Embeddedness: Being an embedded system, a CES is usually subject to stringent timing requirements. For automotive applications, the variability in timing is usually bounded by a range of milliseconds, whereas for the transport robot use case deadlines are given in seconds and originate from the CSG's context, for example, manufacturing execution system (MES) execution cycles. If the systems repeatedly fail to adhere to the timing requirements, the faults can accumulate and ultimately cause a fleet failure. Another consequence of acting in the physical world and, more precisely, of being connected via a wireless network, is the possibility of message loss. Finally, embedded systems have limited computational resources and are often powered by battery. Thus, implementations must be efficient and the number of messages exchanged for negotiation between CESs, as well as for communication between monitors, should be minimal.

Runtime Monitoring of Interaction Protocols

In this section, we provide an example of an interaction protocol of the transport robot use case, which serves as the subject for our runtime monitoring approach. We then introduce two specification formalisms, each targeting different aspects of the challenges identified for runtime monitoring of CSGs and give a high-level description of how to apply them to the example introduced.

Figure 10-8 shows an Agent UML (AUML) [Cabac et al. 2004] sequence diagram of the distributed order assignment, an auction-based algorithm, used to assign transport jobs in the transport robot use case. AUML is a natural fit for the description of interaction protocols because it is widespread, relatively easy to use, and can serve as a semi-formal development artifact at every stage of the system design process.

The protocol is initiated whenever a machine broadcasts the need for transport to the fleet. Two general things should be noted here. First, a protocol deadline of 120 seconds is specified in the top left corner to ensure the (timely) termination of the protocol. Second, we use different execution lines for the CES and CSG, yet the former is by definition a member of the latter. This is necessary to model the perspective of a CES, where a monitor ultimately resides. Initially, the MES addresses the entire CSG via a broadcast message, represented by an empty circle arrowhead. After the announcement is received, all robots will wait two seconds before continuing with the protocol,

www.dbooks.org

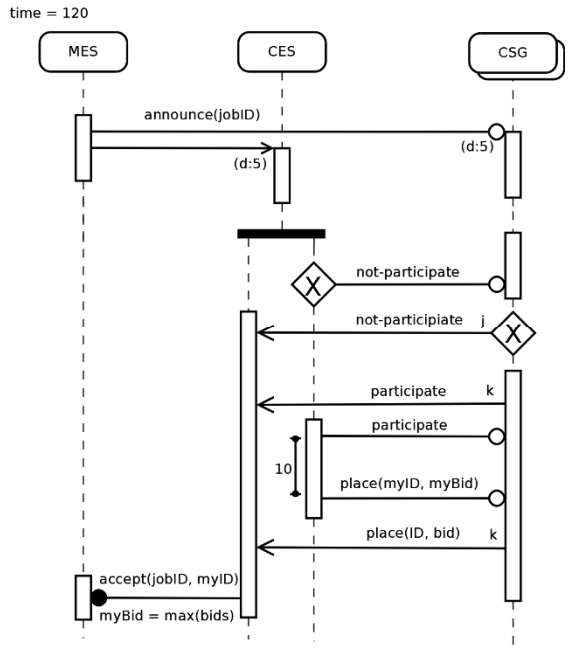


***Fig. 10-8**: An AUML diagram of distributed order assignment in the autonomous transport robots use case*

which is specified as (d:5) under the message. At this point, two concurrent threads (parallel vertical bars) are run per robot: one for sending messages and one for receiving messages. This way, no false assumptions about the order of events are incorporated into the model. The robot will then continue to inform the fleet about its readiness to participate in the current auction. A diamond box with a cross represents an "exclusive or" decision — that is, a robot should only ever send one of the two messages. Every other member of the CSG makes an analogous decision. All participating units then calculate their bids in a subroutine (which is not shown in the diagram) and notify the fleet again via broadcast. Each CES announces its bid via broadcast message and waits for all other bids to arrive, with the same number of bids as participation announcements expected in total. The bids of all participating CESs should be received within 10 seconds, which is represented by the vertical line on the right-hand side of the figure. The winner is determined using the bids received, where the robot with the highest bid wins; IDs can be used for symmetry breaking. The black circular arrowhead indicates that the winning CES will then notify the machine with a reliable message that is sent until it has been acknowledged.

Monitoring Functional Correctness

Certifying distributed algorithms are a distributed runtime monitoring technique [Voellinger and Akili 2018]. For its (distributed) input-output pair (i, o), a certifying distributed algorithm (CDA) computes, in addition to the output o, a witness w. A witness is an object which can be used in a formal argument for the correctness of the input-output pair. A witness predicate Γ holds for the triple (i, o, w) if the pair (i, o) is correct. The witness predicate is decided by a distributed checker algorithm at runtime. The idea is that a user of a CDA does not have to trust the actual algorithm but rather the checker, which is simpler and can be formally verified. Using the terminology of runtime verification, a checker acts as a monitor for a system running a CDA. The system itself is instrumented to additionally compute a witness.
CDAs can be used to verify functional correctness at runtime. With respect to the distributed order assignment (Figure 10-8), we identified the following functional specification:

- **Agreement:** All robots agree on the winner triple (winnerID, winner bid, jobID)
- **Existence:** There is a robot with the winnerID
- **Maximum:** The winner's bid is maximal among all bids

For a robot *k*, we consider its unique identifier as input ($i_k := \{k\}$) and a triple containing the ID of its determined winner, the bid of its determined winner, and job ID as local output ($o_k := \{(winnerID_k, winnerBid_k, jobID_k)\}$. The witness of robot *k* consists of its own bid as well as a set containing the outputs of all other robots ($w_k := (bid_k, \{o_l \mid l \in ID \text{ and } l \neq k\})$.

We distinguish between input, output and witness of single robots and those of the whole CSG. We denote the latter as global input I, global output O and global witness W, and define these as the union of the corresponding local items of all robots.

We formalize the specification as the three global predicates Γ_{agree}, Γ_{exist}, Γ_{max} over the global input, output, and witness.

If Γ_{agree} holds for (I, O, W), then the property *agreement* holds. For each of the global predicates, we introduce a local predicate that can be checked by a monitor for each robot: γ_{agree}, γ_{exist}, γ_{max}. We forgo the formalization of the predicates but only state their meaning.

The local predicate γ_{agree} holds for robot *k* if its winner triple equals the winner triple of all other robots. If γ_{agree} holds for all robots, Γ_{agree} holds for the CSG. The predicate γ_{max} holds for a robot if its bid is less than or equal to its winner bid. The predicate γ_{max} must hold for all

www.dbooks.org

robots. However, note that this predicate would hold for all robots even if each robot had a different *winnerBid* to compare its bid with. To verify the maximum among all bids, each robot has to compare its bid with the same winner bid. However, with γ_{agree} holding for all robots, this is ensured. Hence, if γ_{max} and γ_{agree} hold for all robots, Γ_{agree} holds for the CSG. The predicate γ_{exist} holds for a robot $\underline{k}$ if its *ID* and *bid* equals its *winner-ID* and *-bid*, that is, if *k* chooses itself as a winner. There must be one robot for which γ_{exist} holds. Together with γ_{agree} holding for all robots, this ensures that there is exactly one winner. Hence, if γ_{exist} and γ_{agree} hold for all robots, Γ_{exist} holds for the CSG.

The monitor of a robot *k* must communicate with the monitors of all other robots in order to collect their outputs, which are contained in w_k. Based on (i_k, o_k, w_k), the monitor of a robot evaluates γ_{agree}, γ_{exist}, γ_{max} based on its robot input, output, and witness. To decide Γ_{agree}, Γ_{exist} and Γ_{max}, the monitors have to combine their results, for example, using a spanning tree as communication topology. To ensure the correctness of the result, a reliable message passing mechanism such as remote procedure call must be used for this exchange.

Monitoring Correct Timing Behavior

Temporal logics are widely employed in the field of runtime monitoring to specify system properties [Bauer et al., 2011]. A well-established specification language for monitoring is Metric Temporal Logic (MTL), which enriches the temporal operators □ (always), ◇ (sometime), and **U** (until) with quantitative timing constraints. The syntax of MTL is given by:

$$\varphi ::= \bot \mid p \mid (\varphi \rightarrow \psi) \mid (\varphi \, \mathbf{U}_t \, \psi)$$

The until operator has a scalar constraint $t \in]0, \infty[$, which intuitively corresponds to a deadline. Other operators can be defined as usual: $\neg\varphi := (\varphi \rightarrow \bot)$, $\top := \neg\bot$, $(\varphi \vee \psi) := (\neg\varphi \rightarrow \psi)$, $(\varphi \wedge \psi) := \neg(\neg\varphi \vee \neg\psi)$, $(\varphi \oplus \psi) := ((\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi))$, $\Diamond_t \varphi := (\top \, \mathbf{U}_t \, \varphi)$, $\Box_t \varphi := \neg\Diamond_t \neg\varphi$, etc. In order to define the semantics of an MTL formula with respect to some SUM, the SUM is instrumented to produce a trace of timestamped events $\rho = (\tau_1, \sigma_1), (\tau_2, \sigma_2), ..., (\tau_n, \sigma_n) \in (\mathbb{R}^{\geq 0} \times \Sigma)^*$ over a finite alphabet Σ. The length of a trace is denoted as $|\rho|$. The semantics of ⊥, p, and → is defined as in classical Boolean logic. For example, $(\rho, i) \models (\varphi \rightarrow \psi)$ if $(\rho, i) \models \varphi$ implies $(\rho, i) \vdash \psi$. The semantics of the until operator $\mathbf{U}_t$ is as follows:

$(\rho, i) \models (\varphi \, \mathbf{U}_t \, \psi)$ if there exists a j such that
$i < j < |\rho|$, $(\rho, j) \models \psi$, $\tau_j - \tau_i \leq t$,
and $(\rho, k) \models \varphi$ for all k with $i < k < j$

In other words, ψ must be true some time before the deadline t has been passed and before that, φ has to continually hold.

With respect to the protocol presented, the following formula expresses that within five seconds after receiving the announce message, each robot declares its participation or non-participation in the bidding:

$$\varphi_1 = (\text{announce} \rightarrow (\Box_5 \neg(\text{participate} \oplus \text{not-participate})))$$

Analogously, the following formula expresses the 10 second timeout for placing a bid:

$$\varphi_2 = (\text{participate} \rightarrow \Box_{10}\, \text{bid})$$

One such monitor checking the formulas above runs for each robot. Thus, the method is implicitly constrained to specify properties of the actions and observations of a single robot.

The Boolean semantics of MTL given above has been extended to a real-valued semantics, where the truth value of a formula is a real number (where ∞ represents *true* and -∞ *false*) [Dokhanchi et al. 2014]. This value gives the robustness of validity or falsity of a formula φ: If φ evaluates to the positive robustness ε, then the specification is true and, moreover, the trace can tolerate perturbations up to ε and still satisfy the specification. Similarly, if the robustness is negative, then the specification is false and, moreover, the trace under ε perturbations still do not satisfy it. This is useful for monitoring, e.g., properties such as "If a town sign is detected, within 3 seconds, the speed is reduced to 50 km/h", which is formulated as

$$(\text{town-sign} \rightarrow \Diamond_3\, (\text{speed}<50))$$

In each timed event, the truth value of the basic event (speed<50) could depend on the value of the actual speed minus 50, thus a trace where the speed is reduced to 40 km/h has a higher robustness value than one where it is reduced only to 49 km/h.

In [Lorenz and Schlingloff 2018], we use a similar idea, however, instead of giving a fuzzy semantics to basic propositions, we let the truth value reflect the robustness with which deadlines are met. In our logic RVTL, the truth value of a formula with respect to a finite trace depends on the distance between the end of the trace and the bounds of the temporal operators in the formula. Formally,

$(\rho,i)[\![\Diamond_t\, \varphi]\!] = (\tau_i+t) - \tau_n$, if $(\tau_i+t)\geq\tau_n$ and $(\rho,k)[\![\Diamond_t\, \varphi]\!]<\infty$ for all $i\leq k\leq n$, and $(\rho,i)[\![\Diamond_t\, \varphi]\!] = \inf\{(\rho,j)[\![\varphi]\!] \mid (\tau_i+t)\geq\tau_j\}$, else.

www.dbooks.org

Intuitively, if the deadline extends past the end of the trace and φ is not satisfied until then, the truth value of $\Diamond t\ \varphi$ reflects how much time is left to satisfy φ. Otherwise, the truth value coincides with the classical meaning in MTL. Therefore, the value $(\rho,i)[\![\Diamond_t\ \varphi]\!]$ provides runtime information about the distance between the current time step and the deadline t for φ. It quantifies how much time is left for φ to become true before its deadline is surpassed. The value of the dual formula $(\rho,i)[\![\Box t\ \varphi]\!]$ is calculated similarly:

$(\rho,i)[\![\Box_t\ \varphi]\!] = \tau_n - (\tau_i+t)$, if $(\tau_i+t)\geq\tau_n$ and $(\rho,k)[\![\Box_t\ \varphi]\!] > -\infty$ for all $i\leq k\leq n$, and $(\rho,i)[\![\Box_t\ \varphi]\!] = \sup\{(\rho,j)[\![\varphi]\!] \mid (\tau_i+t)\geq\tau_j\}$, else.

That is, if the deadline extends past the end of the trace, then the truth value of $\Box_t\ \varphi$ reflects the "obligation" to obey φ for some prolonged time; otherwise, the truth value coincides with the classical meaning. With such a semantics, we can issue a warning already if deadlines are nearly missed, even before an error occurred. A typical formula is

$$\varphi_3 = (\text{orderCreated} \rightarrow \Diamond_{600}\ \text{orderCompleted})$$

which states that every transport job should be completed within ten minutes. Monitoring this formula for several days in a real production environment shows situations where "near misses" accumulate more and more, until finally "real misses" of the deadline occur. In a collaborative work environment, such an agglomeration of problems can be an early indication that the size of the fleet needs to be increased.

10.5 Conclusion

In this chapter, we elaborated on a notion of trust in the context of collaborative embedded systems. We discussed how different aspects of trust can be addressed at design time and runtime. During design time, testing the behavior of collaboration functions in an extended set of test scenarios creates trust by enabling software behavior certification. During design time, the prediction of software and system behavior gives insights into decisions. In the case of dangerous predictions, failover behavior can be triggered. We then presented runtime monitoring — a lightweight method for establishing trust of a user in a CSG. To this end, we introduced two runtime monitoring techniques: certifying distributed algorithms and runtime verification with temporal logics. Certifying distributed algorithms are tailored for distributed runtime monitoring and therefore well-suited for application to non-intermediate interaction through negotiation

protocols. The method supports distribution of a specification for the global behavior of the system in a way that partial specifications can be checked locally at each component. Temporal logics, on the other hand, are a good fit to address the challenges posed by the physical embedding of a CES. They can be used to express the timing of behaviors as typically required for embedded systems. Moreover, multi-valued variants of linear temporal logic can even help to detect progressing fault chains before they lead to failures.

10.6 Literature

[ANKI 2020] Overdrive – https://anki.com/en-us/overdrive.html; accessed on 07/14/2020.

[Avizienis et al. 2004] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing. In: IEEE Transactions on Dependable and Secure Computing, 2004, pp.11-33.

[Bartocci et al. 2018] E. Bartocci, Y. Falcone, A. Francalanza, G. Reger: Introduction to Runtime Verification. In: Lectures on Runtime Verification, 2018, pp. 1-33.

[Bauer et al. 2011] A. Bauer, M. Leucker, C. Schallhart: Runtime Verification for LTL and TLTL. In: ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, pp. 1-64.

[Blockly 2020] Google Blockly – https://developers.google.com/blockly; accessed on 07/14/2020.

[Cabac et al. 2004] L. Cabac, D. Moldt: Formal Semantics for AUML Agent Interaction Protocol Diagrams. In: International Workshop on Agent-Oriented Software Engineering, 2004, pp. 47-61.

[da Silva Amorim et al. 2016] S. da Silva Amorim, J. D. McGregor, E. S. de Almeida, C. von Flach, G Chavez: Software Ecosystems Architectural Health: Challenges x Practices. In: Proceedings of the 10th ECSA Workshops. ACM, 2016, pp. 1-7.

[da Silva Amorim et al. 2017] S. da Silva Amorim. F. S. S. Neto, J. D. McGregor, E. S. de Almeida, C. von Flach, G Chavez: How Has the Health of Software Ecosystems Been Evaluated?: A Systematic Review. In: Proceedings of the 31st Brazilian Symposium on Software Engineering. ACM, 2017, pp. 14–23.

[Dokhanchi et al. 2014] A. Dokhanchi, B. Hoxha, G.s Fainekos: On-Line Monitoring for Temporal Logic Robustness. 5th International workshop on Runtime Verification (RV 2014), Toronto. Springer LNCS 8734, 2014, pp. 231-246.

[Kephart and Chess, 2003] J. O. Kephart, D. M. Chess: The Vision of Autonomic Computing. Computer, vol. 36, no. 1, pp. 41–50, 2003.

[Krasner and Pope 1988] G. Krasner, S. Pope: A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80. In: Journal of Object-Oriented Programming.

[Kuhn et al. 2013] T. Kuhn, T. Forster, T. Braun, R. Gotzhein: Feral — Framework for Simulator Coupling on Requirements and Architecture Level. In: Formal Methods

www.dbooks.org

and Models for Codesign (MEMOCODE), 2013 EleventhIEEE/ACM International Conference on. IEEE, 2013, pp. 11–22.

[Leucker and Schallhart 2009] M. Leucker, C. Schallhart: A Brief Account of Runtime Verification. In: The Journal of Logic and Algebraic Programming, Vol. 78 Issue 5, 2009, pp. 293-303.

[Lorenz and Schlingloff 2018] F. Lorenz, H. Schlingloff: Online-Monitoring Autonomous Transport Robots with an R-valued Temporal Logic. 14th International IEEE Conference on Automation Science and Engineering (CASE), 2018.

[Luckcuck et al. 2019] M. Luckcuck, M. Farrel, L. Dennis, C. Dixon, M. Fisher: Formal Specification and Verification of Autonomous Robotic Systems: A Survey. In: ACM Computing Surveys (CSUR), 2019, pp.1-41.

[Voellinger and Akili 2018] K. Völlinger, S. Akili: On a Verification Framework for Certifying Distributed Algorithms: Distributed Checking and Consistency. In: International Conference on Formal Techniques for Distributed Objects, Components, and Systems, 2018, pp. 161-180.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Check for updates

Arvid Butting, RWTH Aachen University
Andreas Wortmann, RWTH Aachen University

11

Language Engineering for Heterogeneous Collaborative Embedded Systems

At the core of model-driven development (MDD) of collaborative embedded systems (CESs) are models that realize the different participating stakeholders' views of the systems. For CESs, these views contain various models to represent requirements, logical functions, collaboration functions, and technical realizations. To enable automated processing, these models must conform to modeling languages. Domain-specific languages (DSLs) that leverage concepts and terminology established by the stakeholders are key to their success. The variety of domains in which CESs are applied has led to a magnitude of different DSLs. These are manually engineered, composed, and customized for different applications, a process which is costly and error-prone. We present an approach for engineering independent language components and composing these using systematic composition operators. To support structured reuse of language components, we further present a methodology for building up product lines of such language components. This fosters engineering of collaborative embedded systems with modeling techniques tailored to each application.

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_11

www.dbooks.org

11.1 Introduction

Collaborative embedded systems

Engineering collaborative embedded systems (CESs) and collaborative system groups (CSGs) usually demands the cooperation of experts from various domains with different backgrounds, methods, and solution paradigms that contribute to different viewpoints (e.g., requirements, functional, logical, or technical viewpoints) of the system [Pohl et al. 2012].

The need to translate domain-specific solution concepts into software artifacts introduces a conceptual gap between the experts' problem domains and the solution domain of software engineering. This gap can give rise to accidental complexities [France and Rumpe 2007] due to the mismatch of solving problem domain challenges with solution domain (programming) concepts.

Model-driven development

Model-driven development (MDD) [France and Rumpe 2007] is a software engineering paradigm that lifts models to the primary development artifacts. In contrast to program code, which reifies concepts of the solution domain, models can leverage domain-specific concepts and terminology to express concepts of the problem domain, which facilitates contribution by domain experts. Models can also be more abstract and leave implementation details to smart software engineering tools (e.g., model transformations or code generators).

To enable models to be processed automatically, they must conform to explicit modeling languages [Hölldobler et al. 2018]. Engineering modeling languages is a challenging endeavor due to the multitude of formalisms and technologies involved, such as (i) grammars [Hölldobler and Rumpe 2017] or metamodels [Eysholdt et al. 2009] to define the languages' syntax, (ii) the Object Constraint Language (OCL) [Cabot and Gogolla 2012] or programming languages to define their well-formedness, and (iii) code generators [Kelly and Tolvanen 2008] or model transformations [Mens and van Gorp 2006] to realize their semantics (in the sense of meaning [Harel and Rumpe 2004]). As "software languages are software too" [Favre 2005], they are also subject to all the challenges typical to complex software as well. And similar to general software engineering, reuse is also the key to the efficient engineering of modeling languages. This holds especially for engineering collaborative embedded systems under the contribution of domain experts through viewpoints that are realized via domain-specific languages.

Software language engineering

Software language engineering (SLE) [Hölldobler et al. 2018] is a field of research that investigates the engineering, maintenance, evolution, and reuse of software languages. Research in SLE has produced a variety of solutions for reusing languages and language parts. However, the approaches for reusing complete (comprising realizations of syntax and semantics) language parts are missing, which severely hampers modeling for CESs and CSGs.

To address this, we present a method for modularizing modeling languages as language components, composing these, and ultimately building product lines of modeling languages to increase the reuse of languages beyond clone-and-own [Dubinsky et al. 2013].

Example 11-1: A family of architecture description languages

Consider a company that develops software for various kinds of CESs that operate in a smart factory. The company employs an architecture description language (ADL) [Medvidovic and Taylor 2000] to develop software component models for the software architecture of the CESs. The different kinds of CESs yield particularities regarding their software architecture. For some systems, it should be possible to perform dynamic reconfiguration of their software architecture based on mode automata [Butting et al. 2017], while for other systems, this is not allowed due to security restrictions. Similarly, some systems support dynamic re-deployment of software components to other systems, while this is not intended for other systems. To reify this properly in the models, the company uses different variants of ADLs — that is, variants of logical and technical viewpoints [Pohl et al. 2012]. These variants have several common language concepts and share large parts of the code generators employed. Without proper language modularization and reuse, these language variants co-exist in the form of cloned-and-owned, monolithic software tools.

In the following, Section 11.2 introduces the MontiCore language workbench, which our solution builds upon. Section 11.3 then introduces our notion of language components, before Section 11.4 explains their composition. Section 11.5 explains how we leverage composable language components to structure language reuse through explicit variability models, which we employed in CrESt to develop variants [Butting et al. 2019] of the MontiArc ADL [Haber et al. 2012] tailored to the use cases of “Autonomous Transport Robots” and “Adaptable and Flexible Factory” (cf. Chapter 1). Section 11.6 concludes this chapter.

www.dbooks.org

11.2 MontiCore

MontiCore [Hölldobler and Rumpe 2017] is a language workbench [Erdweg et al. 2015] that facilitates the engineering of compositional modeling languages. MontiCore languages are based on a context-free grammar (CFG) that defines the (concrete and abstract) syntax of the respective language to which its models must conform. MontiCore uses this CFG to generate a parser that can process models of that language, along with abstract syntax classes that can store the machine-processable representation of the models once they have been parsed.

Abstract syntax tree

After parsing, the models are translated into abstract syntax trees (ASTs) — that is, instances of the abstract syntax classes generated from the grammar. Using MontiCore's extensional function library, these models are checked for well-formedness and other properties, transformed, and ultimately translated into other models, reports, source code, or other target representations. All of these activities rely on MontiCore's modular visitors that process parts of the AST. Visitors

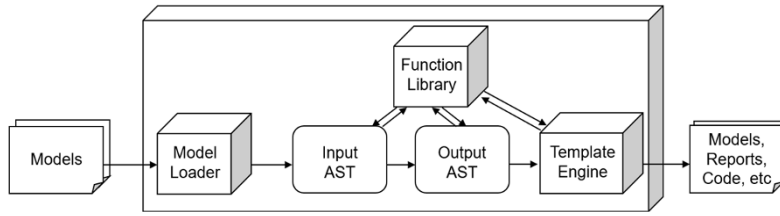


***Fig. 11–2**: The quintessential components of MontiCore's language processing tool chain support model loading, checking, and transformation*

[Gamma et al. 1995] separate operations on object structures from the object structures themselves and thus enable the addition of further operations without requiring modifications to the object structures.

Symbols

To facilitate operation on different nodes of the AST, MontiCore supports the definition of symbols—meaningfully abstracted model parts—based on grammar rules. Symbols are stored in symbol tables and can be resolved within a language as well as by other languages, enabling different forms of language composition.

Using CFGs and symbol tables, MontiCore supports the modular composition of languages through extension, embedding, and aggregation: language extension enables a CFG to extend another CFG, thereby inheriting all productions of the extended CFG. This process produces a new AST that may reuse productions of the extended CFG. This is useful, for example, for extending a base language in different ways with domain-specific extensions that would otherwise

convolute the base. Language embedding is the integration of selected productions of the client CFG into extension points of the host CFG. The resulting AST is the AST of the host CFG with a sub-AST of the client CFG embedded into selected nodes. This supports the creation of (incomplete) languages that provide an overall structure but demand (domain-specific) extension. Language aggregation is the integration of languages through references between their modeling elements. These references are resolved using MontiCore's symbol table framework and do not yield integrated ASTs. Instead, the models of the integrated languages remain separate artifacts. This supports, for example, the separation of different, yet integrated, concerns in models, such as structure and behavior.

For well-formedness checking and code generation, MontiCore provides generic infrastructures that can be customized by adding well-formedness rules (context conditions) and FreeMarker [Forsythe 2013] templates that define the code generation by processing the AST using template control structures and target language text. Consequently, a MontiCore language usually comprises a CFG, context conditions, and FreeMarker templates.

11.3 Language Components

Component-based software engineering is a paradigm for increasing software reusability by means of modularization. This paradigm is successfully applied in different domains and well suited for the engineering of embedded systems. The techniques of this paradigm can be applied to software languages as well. As a consequence, all advantages of component-based software engineering, such as increased reusability and better maintainability, can be leveraged to facilitate SLE. Similar to [Clark et al. 2015], we use the term *language component* for modular, composable software language realizations.

Definition 11-3*: Language component*

A language component is a reusable unit encapsulating a potentially incomplete language definition. A language definition comprises the realization of syntax and semantics of a (software) language.

This definition reduces the notion of language components to the constituents of the language infrastructure without being dependent on a specific technological space [Kurtev et al. 2002]. Ultimately, this means that a language component is a set of artifacts that form a

www.dbooks.org

reusable unit. This set includes both handwritten as well as generated artifacts of language-processing tooling. For textual languages, it may include, for example, a grammar as a description of the syntax, the source code realizing well-formedness rules, a generated parser, and a generated AST data structure. In other technological spaces, a language component may contain a metamodel instead of a grammar and parser. Some language workbenches, such as MontiCore, enable language engineers to customize generated artifacts. Such handwritten customizations are part of a language component as well.

Extension points

Ideally, software components are black boxes whose internal workings are not relevant in their environment [McIlroy 1968]. Consequently, language components may also hide implementation details from their environment. To this end, language engineers can plan explicit extension points of a language component for which other language components can provide extensions. The realization of the extension points and extensions depends on the technological space used to realize the language components. In MontiCore, for example, syntax extension points can be realized through underspecification in grammars realized as interfaces or external productions [Hölldobler and Rumpe 2017]. Other language constituents, such as code generators, may yield different mechanisms for extension points and extensions.

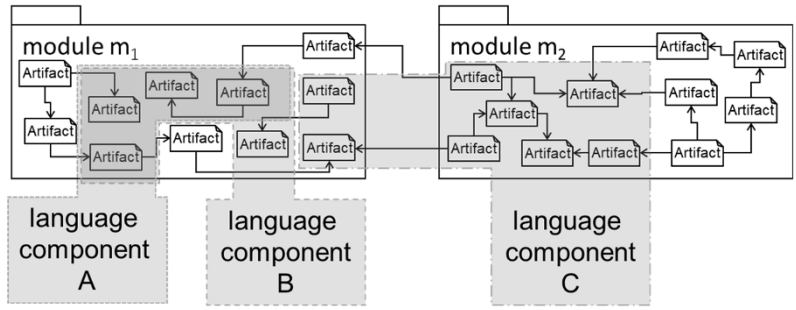


Fig. 11–4: *Artifacts of a language component can be distributed among software modules and some artifacts belong to multiple language components*

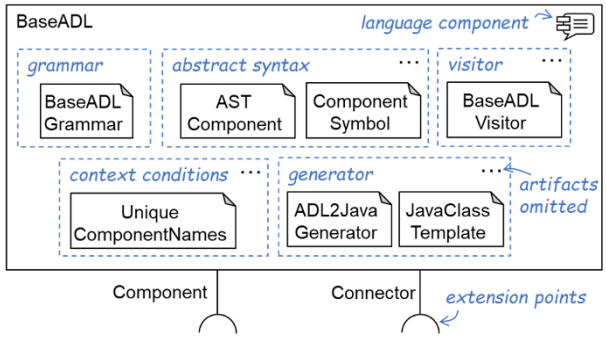
Artifact organization

A language component consists of many interrelated artifacts that may be distributed across different software modules and a single software module may contain artifacts for one or more language components (cf. Figure 11–4). This is due to the fact that the modularization of software into modules is typically driven by build tools (e.g., Maven or Gradle) that intend a different level of granularity.

Furthermore, an artifact may be part of multiple language components.

Example 11-5: BaseADL language component in MontiCore

The BaseADL language component contains a context-free grammar to describe the concrete and abstract syntax of a basic architecture description language (ADL). From this grammar, MontiCore generates a set of AST and symbol table classes that represent the abstract syntax data structure, a parser, a visitor infrastructure, and an infrastructure for realizing and checking context conditions. The handwritten context conditions, code generator classes, and templates are part of the language component as well.



In this example, the language engineers have planned two extension points for the BaseADL language component. One extension point can be extended to introduce a new notation for components and another one to introduce a new kind of connector. The extension point for components, for example, can be extended to add dynamic components that contain a mode automaton (cf. Example 11-1).

To identify, analyze, compose, and distribute language components, the large number of source code artifacts that realize the language component have to be extracted from the software modules. The constituents of a language component can be described and typed through a suitable *artifact model* [Butting et al. 2018b]. This produces the opportunity to identify the constituents of a language component by means of an artifact data extractor in a semi-automated process. This process collects potential artifacts of a language component, starting with a central artifact such as a grammar or a metamodel. With an underlying artifact model, an artifact data extractor can extract all associations from this artifact to other artifacts. For instance, in the technological space of MontiCore, this automated

www.dbooks.org

extraction handles the identification of all Java classes that realize context conditions that can be checked against abstract syntax classes generated from a grammar.

However, the result of this automatic extraction (1) can produce artifacts that are not intended to be part of a language component or (2) can lack artifacts intended to be part of the language component. Therefore, handwritten adjustments of this result must be considered. In other technological spaces, these data extractors must be provided accordingly.

11.4 Language Component Composition

Forms of language composition

In general, the engineering of language components as described in Section 11.3 is the basis for building languages by composing language components. There are various forms of language

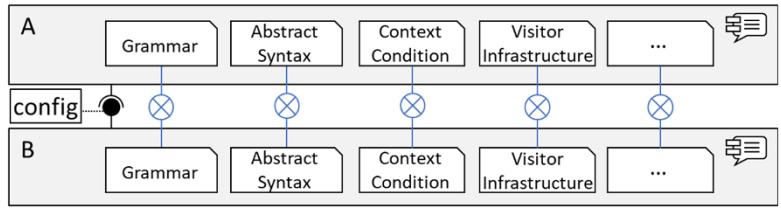


Fig. 11–6: *Composing two language components A and B requires composition of their constituents*

composition [Erdweg et al. 2012] that are supported by different language workbenches [Méndez-Acuña et al. 2016]. Some forms of language composition produce composed languages that can process integrated model artifacts, while other forms—such as language aggregation—integrate languages whose models remain in individual artifacts. Certain kinds of language composition—for example, language extension and language inheritance—require that one language depends on another language. These forms are not suitable for independent engineering of the participating languages and, when applied to language components, may introduce dependencies to the language component context. Some forms of language composition also require configuration with integration "glue," such as adapters between two kinds of symbols [Nazari 2017]. Therefore, care must be taken to select a suitable form of language composition.

Language component composition operators

For the composition of language components, we generalize the concrete form of language composition and denote that each composition of two language components is specified through a

configuration, as depicted in Figure 11–6. The configuration connects an extension point of a language component with an extension of another language component and states which form of composition has to be applied. Depending on the form of composition, the composition may also have to be configured with glue code. The actual composition of two language components is realized through the composition of their constituents. To this end, composition operators must be defined for each kind of constituent individually.

For example, MontiCore enables the composition of language components through embedding. The actual embedding has to be performed for handwritten constituents—such as grammars, context conditions, and generators—but also for generated constituents such as the AST data structures, the symbol table, and the visitor infrastructure. Thus, for all these constituents, an individual composition operator that realizes the embedding must be defined.

Composing grammars

MontiCore enables grammars to inherit from one or more other grammars. If a grammar inherits from another (super-)grammar, it can reuse and, optionally, extend or override the productions of the super-grammar. This influences the syntax through the generated parser and the integrated AST infrastructure, but also affects many other parts of the language-processing infrastructure generated from a grammar. Multi-inheritance in grammars can be used to compose two independently developed grammars and through this, realize language embedding. Therefore, the composition operator for embedding a MontiCore grammar into another MontiCore grammar produces a new grammar that inherits from both source grammars [Butting et al. 2019]. Furthermore, a grammar production integrating extension point and extension are generated, depending on the kind of syntax extension point (e.g., an interface production) and the kind of extension (e.g., a parser production).

Composing context conditions

In the context of language composition, we distinguish between *intra-language* and *inter-language* context conditions. Intra-language context conditions check the well-formedness of the syntax of a single language component, while inter-language context conditions affect syntax elements of more than one language component. Intra-language context conditions are part of a language component, whereas we regard inter-language context conditions as part of the configuration of the composition. Context conditions in MontiCore are evaluated against the abstract syntax by means of a visitor. To this end, composing context conditions of different language components requires the composition of the underlying visitor infrastructures. This is realized via inheritance and delegator visitors [Heim et al.

www.dbooks.org

2016]. Once the visitors are integrated, the context conditions can be checked against the integrated structure.

Composing code generators

Code generators are commonly used for translating models into implementations that can be executed on embedded systems. However, few techniques for the composition of code generators exist, and these rarely enable composition of independent code generators. Code generator composition is challenging, as the result of the composition should produce correct code. While this is generally impossible, we can support language engineers in developing code generators that produce code that is structurally compatible with code generated by other code generators [Butting et al. 2018a]. This is realized by requiring each generator to indicate an *artifact interface* to which the generated code conforms. An adapter resolves potential conflicts between the artifact interfaces of two different code generators.

A further challenge in code generator composition is the coordination of the code generator execution. For some forms of composition, such as language embedding, code generators have to exchange information and thus comply with each other in a similar way to the generated code. To this end, generators provide *generator interfaces* to which the code generators conform. Again, potential conflicts between two code generators that are to be composed are resolved via adapters.

11.5 Language Product Lines

Reuse of languages or language parts is not only beneficial for language engineers due to the decreasing development cost and the increase in the language tooling quality, but also for language users, as the accidental complexity [Brooks 1987] posed by the effort of learning the syntax of new languages is reduced. In the context of engineering CESs and CSGs, language product lines are very applicable. Despite the variety in fields of application for which CESs and CSGs are employed, their model-driven engineering often relies on the same general-purpose modeling languages (e.g., UML) to describe aspects such as the geometry of physical entities of CESs, their system functions, collaboration functions, their communication paradigms, architectures, goals, capabilities, and much more.

This raises a gap between the problems in the application domain and the ability to express these in the modeling languages in a compact and understandable way. Enriching general purpose

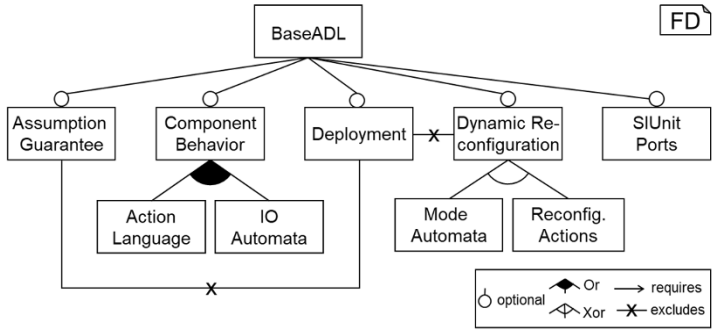
modeling language with application domain-specific language concepts helps to bridge this gap. Modular language engineering in terms of developing language components as presented in Section 11.3 and composing these as presented in Section 11.4 can be used to realize product lines of languages [Butting et al. 2019]. Such *language product lines* enable systematic reuse of language components for a family of similar languages and, therefore, enable individual tailoring of the modeling languages to the application fields of CESs and CSGs.

Modeling language product lines

The variability of the language product line in terms of language features is modeled as a *feature diagram*, where language features are realized as language components. Therefore, a *binding* of the product line connects features with the language components that realize them. Furthermore, the binding configures the pairwise language component compositions that occur in all products of the language product line.

Example 11-7: MyADL language product line

The company developing CESs described in Example 11-1 can employ a language product line for their ADLs to eliminate clones of redundant language parts and the resulting effort in maintaining and evolving these individually. All ADL variants have a common base language, and different combinations of extensions to this base language are considered in the product line. The optional behavior of software components can be modeled via input-output automata, an action language, or both. Some application scenarios benefit from using SI units as data types for messages sent via ports.



A product of the product line is specified via a *feature configuration*. The language components of all selected features are composed in pairs, as specified via the binding. The result of composition is a language component. Derivation of languages from

www.dbooks.org

the product line is automated, but the resulting language component can be customized manually (optional). Engineering reusable language components and using these within language product lines fosters separation of concerns among different roles, as depicted in Figure 11–8.

❑ *Language engineers* develop language components and their extension points independently of one another. The artifacts of a language component are identified and collected via an artifact data extractor.

❑ A *product line manager* selects suitable language components for a field of application scenarios, arranges these in the form of a feature model, and configures the composition of the language components in a binding.

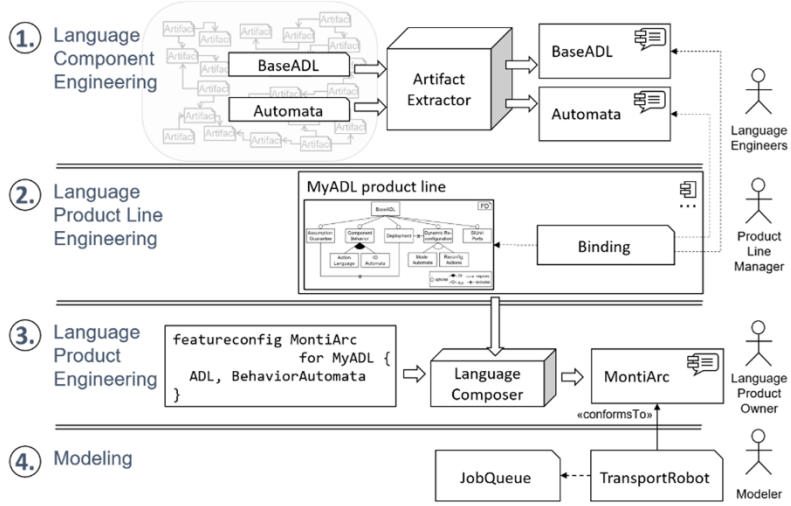


Fig. 11–8: *Processes and stakeholders involved in engineering language product lines*

❑ A *language product owner* selects features of a language product line that are useful for a concrete application and, on a pushbutton basis, can use generated language-processing tools for this language. The generated tooling can be customized (optional). In Figure 11–8, the language product is an ADL with the name "MontiArc."

❑ A *modeler* uses a language product through the generated language-processing tools without being aware of the language product line — for instance, to model specific system functions or collaboration functions of collaborative transport robot systems.

11.6 Conclusion

We have presented concepts for composing modeling languages from tried-and-tested language components. Leveraging these concepts facilitates engineering of the most suitable domain-specific languages for the different stakeholders involved in systems engineering. This mitigates an important barrier in the model-driven development of CESs and CSGs. Future research should encompass generalization of language composition beyond technical spaces and support for language evolution.

11.7 Literature

[Brooks 1987] F. P. Brooks, Jr.: No Silver Bullet: Essence and Accidents of Software Engineering, IEEE Computer (20:4), 1987, pp 10-19.

[Butting et al. 2017] A. Butting, R. Heim, O. Kautz, J. O. Ringert, B. Rumpe, A. Wortmann: A Classification of Dynamic Reconfiguration in Component and Connector Architecture Description Languages. In: Proceedings of MODELS 2017. Workshop ModComp, CEUR 2019, 2017.

[Butting et al. 2018a] A. Butting, R. Eikermann, O. Kautz, B. Rumpe, A. Wortmann: Modeling Language Variability with Reusable Language Components, In: International Conference on Systems and Software Product Line (SPLC'18), 2018, ACM.

[Butting et al. 2018b] A. Butting, T. Greifenberg, B. Rumpe, A. Wortmann: On the Need for Artifact Models in Model-Driven Systems Engineering Projects. In: Software Technologies: Applications and Foundations, Springer, 2018, pp. 146-153.

[Butting et al. 2019] A. Butting, R. Eikermann, O. Kautz, B. Rumpe, A. Wortmann: Systematic Composition of Independent Language Features. In: Journal of Systems and Software, 152, 2019, pp. 50-69.

[Cabot and Gogolla 2012] J. Cabot, M. Gogolla: Object Constraint Language (OCL): A Definitive Guide. In: International School on Formal Methods for the Design of Computer, Communication and Software Systems, Springer, Berlin, Heidelberg, 2012, pp. 58-90.

[Clark et al. 2015] T. Clark, M. v. d. Brand, B. Combemale, B. Rumpe: Conceptual Model of the Globalization for Domain-Specific Languages. In: Globalizing Domain-Specific Languages (Dagstuhl Seminar), LNCS 9400, Springer, 2015, pp. 7-20.

[Dubinsky et al. 2013] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, K. Czarnecki: An Exploratory Study of Cloning in Industrial Software Product Lines. In: Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering, CSMR '13, Washington, DC, USA, 2013, pp. 25–34.

[Erdweg et al. 2012] S. Erdweg, P. G. Giarrusso, T. Rendel: Language Composition Untangled. In: Proceedings of the Twelfth Workshop on Language Descriptions, Tools, and Applications, 2012, pp. 1-8.

www.dbooks.org

[Erdweg et al. 2015] S. Erdweg et al.: Evaluating and Comparing Language Workbenches: Existing Results and Benchmarks for the Future. In: Computer Languages, Systems & Structures 44, 2015, pp. 24-47.

[Eysholdt et al. 2009] M. Eysholdt, S. Frey, W. Hasselbring: EMF Ecore based meta model evolution and model co-evolution. In: Softwaretechnik-Trends 29.2, 2009, pp. 20-21.

[Favre 2005] J. M. Favre: Languages Evolve Too! Changing the Software Time Scale. In: Eighth International Workshop on Principles of Software Evolution (IWPSE'05) IEEE, 2005, pp. 33-42.

[Forsythe 2013] C. Forsythe: Instant FreeMarker Starter. Packt Publishing Ltd, 2013.

[France and Rumpe 2007] R. France, B. Rumpe: Model-Driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering 2007 at ICSE. Minneapolis, IEEE, 2007, pp. 37-54.

[Gamma et al. 1995] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[Haber et al. 2012] A. Haber, J. O. Ringert, B. Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical Report AIB-2012-03, RWTH Aachen University, 2012.

[Harel and Rumpe 2004] D. Harel, B. Rumpe: Meaningful Modeling: What's the Semantics of "Semantics"?. In: IEEE Computer, Volume 37, No. 10, 2004, pp 64-72.

[Heim et al. 2016] R. Heim, P. Mir Seyed Nazari, B. Rumpe, A. Wortmann: Compositional Language Engineering using Generated, Extensible, Static Type-Safe Visitors. In: Conference on Modelling Foundations and Applications (ECMFA'16), LNCS 9764. Springer, July 2016, pp. 67–82.

[Hölldobler and Rumpe 2017] K. Hölldobler, B. Rumpe: MontiCore 5 Language Workbench Edition 2017. In: Aachener Informatik-Berichte, Software Engineering, Band 32. Shaker Verlag, 2017.

[Hölldobler et al. 2018] K. Hölldobler, B. Rumpe, A. Wortmann: Software Language Engineering in the Large: Towards Composing and Deriving Languages. In: Journal of Computer Languages, Systems & Structures, 54, Elsevier, 2018, pp. 386-405.

[Kelly and Tolvanen 2008] S. Kelly, J. P. Tolvanen: Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons, 2008.

[Kurtev et al. 2002] I. Kurtev, J. Bézivin, M. Aksit: Technological Spaces: An Initial Appraisal. In: 4th International Symposium on Distributed Objects and Applications (DOA), 2002.

[McIlroy 1968] M. D. McIlroy: Mass-Produced Software Components, Software Engineering Concepts and Techniques. NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976, pp. 88-98.

[Medvidovic and Taylor 2000] N. Medvidovic, R. N. Taylor: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering 26.1, 2000, pp. 70-93.

[Méndez-Acuña et al. 2016] D. Méndez-Acuña, J. A. Galindo, T. Degueule, B. Combemale, B. Baudry: Leveraging Software Product Lines Engineering in the Development of External DSLs: A Systematic Literature Review. Computer Languages, Systems & Structures, 46, 2016, pp. 206-235.

[Mens and van Gorp 2006] T. Mens, P. Van Gorp: A Taxonomy of Model Transformation. Electronic notes in theoretical computer science 152, 2006, pp. 125-142.

[Nazari 2017] P. Mir Seyed Nazari: MontiCore: Efficient Development of Composed Modeling Language Essentials. In: Aachener Informatik-Berichte, Software Engineering, Band 29. Shaker Verlag, 2017.

[Pohl et al. 2012] K Pohl, H. Hönninger, R. Achatz, M. Broy (Eds.): Model-Based Engineering of Embedded Systems, Springer-Verlag, Berlin Heidelberg, 2012.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Emilia Cioroaica, Fraunhofer IESE
Karsten Albers, INCHRON AG
Wolfgang Boehm, Technical University of Munich
Florian Pudlitz, Technische Universität Berlin
Christian Granrath, RWTH Aachen University
Roland Rosen, Siemens AG
Jan Christoph Wehrstedt, Siemens AG

12

Development and Evaluation of Collaborative Embedded Systems using Simulation

Embedded systems are increasingly equipped with open interfaces that enable communication and collaboration with other embedded systems, thus forming collaborative embedded systems (CESs). This new class of embedded systems, capable of collaborating with each other, is planned at design time and forms collaborative system groups (CSGs) at runtime. When they are part of a collaboration, systems can negotiate tactical goals, with the aim of achieving higher level strategic goals that cannot be achieved otherwise. The design and operation of CESs face specific challenges, such as operation in an open context that dynamically changes in ways that cannot be predicted at design time, collaborations with systems that dynamically change their behavior during runtime, and much more. In this new perspective, simulation techniques are crucially important to support testing and evaluation in unknown environments. In this chapter, we present a set of challenges that the design, testing, and operation of CESs face, and we provide an overview of simulation methods that address those specific challenges.

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_12

12.1 Introduction

Modeling and simulation are established scientific and industrial methods to support system designers, system architects, engineers, and operators of several disciplines in their work during the system life cycle. Simulation methods can be used to address the specific challenges that arise with the development and operation of collaborative embedded systems (CESs). In particular, the evaluation of collaborative system behavior in multiple, complex contexts, most of them unknown at design time, can benefit from simulation. In this chapter, after a short motivation, we exemplify scenarios where simulation methods can support the design and the operation of CESs and we summarize specific simulation challenges. We then describe some core simulation techniques that form the basis for further enhancements addressed in the individual chapters of this book.

12.1.1 Motivation

Simulation is a technique that supports the *overall design, evaluation, and trustworthy operation* of systems in general. CESs are a special class of embedded systems that, although individually designed and developed, can form collaborations to achieve collaborative goals during runtime. This new class of systems faces specific design and development challenges (cf. Chapter 3) that can be addressed with the use of simulation methods.

At *design time,* a suitable simulation allows verification and exploration of the system behavior and the required architecture based on a virtual integration. *At runtime*, when systems operate in open contexts, interact with unknown systems, or activate new[1] system functions, the aspect of *trust* becomes of crucial importance. Using later research and technology advancements, we foresee the possibility of computing trust scores of CESs directly at runtime based on the evaluation results of system behavior in multiple simulated scenarios. The core simulation techniques presented in this chapter form the basis for enhanced testing and evaluation techniques.

[1] "New functions" are functions that have not been enabled before in the current internal system configuration.

www.dbooks.org

12.1.2 Benefits of Using Simulation

Regardless of the domain, the use of simulation methods for behavioral evaluation of systems and system components has multiple benefits.

Simulation to support effectiveness of exploration

For a concrete scenario of complex interactions, simulation methods are more exploratory than analytical methods. The effectiveness of the exploration is achieved through the coupling of detailed simulation models, while the efficiency of the exploration is achieved by exercising a system or system group behavior in a multitude of scenarios, including scenarios that contain failures.

Simulation to evaluate the function interaction

Through the collaboration of CESs, collaborative system groups (CSGs) that did not exist before are formed dynamically at runtime. Moreover, the exact configuration of those CSGs is not known at design time. In such situations, when systems operate in groups that never existed before, there is insufficient knowledge about the collaborative behavior and its effects. In this case, simulation can help to discover the effects of different function interactions.

Closed loop simulation

As a third benefit, the use of closed-loop simulation (X-in-the-loop simulation) is a suitable approach for testing embedded systems (e.g., control units of collaborative assistant systems). The independence of the simulated test environment from the implementation and realization of the embedded system (system under test) generates advantages, such as reusability of the simulations and cost savings in system testing. One example is the testing of different control units—for which the simulation environment can be reused without major adaptions—independently of the implementation and realization concept of the control unit. Only the interfaces of realized functionality of the system under test have to be the same to enable coupling of the simulation and testing environment.

Risk reduction

A fourth major benefit is that the risk for the system user (e.g., car passenger) can be reduced by using simulations during the system testing process by virtual evaluation. The test execution in virtual environments enables discovery of harmful behavior in a virtual world, where only virtual and not real entities are harmed. Real hazards can thus be avoided. In addition, the risk during the operation of collaborative systems can be reduced by using predictive risk assessment by means of simulation.

Virtualization of tests

Additionally, the use of simulations for testing at system design time can be used to make tests virtual, with an associated reduction in hardware and prototypes. In particular, the costs for the production of these real components can be reduced. In addition, making tests virtual leads to early error detection and correction and thus to a

further reduction in development costs. This is especially useful as the exact configuration of CSGs is not known at design time. Here, simulation gives the opportunity to simulate sets of possible (most likely) scenarios.

Reduction of development time and costs

Furthermore, the independence of simulation models that reflect the behavior of real components results in efficient development, because in some use cases, simulations are not bound to real-time conditions. Therefore, they can be executed much faster than in real time and thus be used to reduce development time. It is also easier to explore many more scenarios and variations of scenarios to gain a better overview and trust in the systems.

As a seventh benefit, the use of simulation environments for testing embedded systems is especially independent of external influences of the environment and ensures that tests can be reproduced. This allows efficient tracking and resolution of problems exposed by the simulation and reproduction of the absence of the problems in the updated system configuration.

The last benefit is that the internal behavior of the simulated systems and their visualization are exposed in a broad way. The traceability of the execution of a real system is limited due to hardware and time restrictions. In the simulation, it is easier to log relevant internal system execution and therefore to identify the causes of problems and unexpected behavior.

In the context of developing and evaluating CESs, the use and benefit of simulation—as described above—lie mainly in the first phases of the entire life cycle. In addition, simulation is also used during operation and service—that is, during the runtime of the system. Thus, simulation represents a methodology that can be used seamlessly across all life cycle phases. Accordingly, there are different challenges for simulation as a development methodology and as a validation technique.

12.2 Challenges in Simulating Collaborative Embedded Systems

Even though there are multiple benefits from using simulation, the aspect of simulation for CESs and CSGs poses particular challenges. In this section, we describe the design time and runtime challenges.

www.dbooks.org

12.2.1 Design Time Challenges

To support the use of simulation during the design of collaborative systems, as presented in Chapter 3, multiple challenges must be addressed, as detailed in the following.

One challenge is the *evaluation of function interaction at design time,* because in a simulation of CESs, functions of multiple embedded systems, developed independently, must be integrated to allow evaluation of the resulting system. This is necessary to discover and fix unwanted side effects before the systems are deployed in the real world. Also, the other relevant aspects for the simulation scenario, such as the context or the dynamic behavior of the systems, must be covered. To support this activity, the *integration of different models and tools* is also important. Development of collaborative system behavior relies on simulating models of different embedded systems that are often developed with different tools. Furthermore, the *integration of different simulation models*, sometimes at different levels of detail, represents an important design engineering challenge. This is because the design of CESs relies on the evaluation of collaborative system behavior that can be expressed at different levels of abstraction. Another challenge is the *integration of different aspects of the simulation scenario.* The comprehensive simulation of collaboration scenarios must cover several aspects to achieve a broad coverage of scenarios. Examples are the context of the CSG, the execution platform of thxe systems and the system group, including the functional behavior, the timing behavior, and the physical behavior of the systems and the system group. The different aspects can require dedicated models and must therefore be covered by specialized simulation tools. For a comprehensive simulation of the whole scenario, these models and tools must interact with each other and must be integrated via a co-simulation platform.

Virtual functional test

The use of simulation methods pursues specific strategic goals as well. One of these methods is the *virtual functional test,* which uses simulation to test a certain collaboration functionality or a certain functionality of one system in the collaborating context. The models of the other parts (systems, context, etc.) must include only those details relevant for the functionality being tested.

Virtual integration test

Another purpose of the simulation is the *virtual integration test.* Here, simulation tests the correct collaboration of the different systems or parts of the systems in a virtual environment. The exact structure of the CSG may not be available at design time and can be subject to dynamic changes. Simulation can test multiple scenarios for this structure for a multitude of situations. An early application of

such tests in the design process, before the different systems are fully designed and implemented, will allow early detection of potential problems and hazards for the collaboration behavior.

Design-space exploration

One strategic goal for the application of simulation, especially in early design phases, is to support a *design-space exploration.* The possibility to support the evaluation of a lot of design alternatives and to identify hazards and failures in the different simulation models allows a strategic evolutionary search for a system variant that fulfills the desired goals and requirements.

Fulfillment of requirements

The *determination of fulfilled requirements* allows the simulations to serve as automation tools for test cases. The results must then be linked to the requirements to determine the coverage. Besides the degree of coverage, additional system behavior can be investigated in relation to the requirements. Due to the great complexity of collaborative systems, automated algorithms must be increasingly used. In Section 12.3, we present a possible approach to help developers and testers meet this challenge.

12.2.2 Runtime Challenges

Even though properly tested during design time, CESs face multiple challenges at runtime and the simulation techniques deployed at runtime face particular challenges as well. In this subsection, we list the challenges of CESs and CSGs as introduced in Chapter 2. We then detail the challenges of using simulation to solve these runtime challenges.

Open context

One particular challenge CESs face at runtime is *operation in open contexts*. The external context may change in unpredictable ways during the runtime operation of CESs. In particular, the environment changes and the context of collaboration may change as well. For example, in the automotive domain, a vehicle that is part of a platoon may need to adapt its behavior when the platoon has to reduce the speed due to high traffic. If the vehicle has a strong goal of reaching the target destination at a specific time, it may decide to leave the platoon that is driving at a lower speed and select another route to its destination. For the remaining vehicles within the platoon, the operational context has changed because the vehicle is now no longer part of the platoon and instead, becomes part of the operational context.

The operational context of a CSG may change dynamically as well, either because a CES joins the group or because the CSG has to operate in an environment that was not foreseen at design time. The CSG has

www.dbooks.org

to adapt its behavior in order to cope with the new environmental conditions. For example, a vehicle under the control of a system function in charge of maintaining a certain speed limit within a platoon has difficulty maintaining the speed after it starts raining.

When CESs form at runtime, the runtime activation of system functions poses additional challenges. When the behavior of CESs is coordinated by the collaboration functions that negotiate the goals of the systems and activate system functions, multiple challenges arise when these system functions are activated for the first time. One example is scheduling: the timing behavior of system functions activated for the first time can influence the scheduling behavior of (a) the interacting system functions, (b) the collaboration functions, and (c) of the whole system.

In this case, the functional interaction must be evaluated because when *system functions are activated for the first time,* the way in which they interact with other system functions in specific situations can be faulty.

Runtime evaluation of changing goals

Moreover, *changing goals at runtime* can also have consequences on the CSG or the CESs. In order to form a valid system group, CESs and/or the CSG may need to change their goals at runtime dynamically, which may obviously have significant impact on the system behavior.

Runtime evaluation of changes in the internal structure

The overall *dynamic change of internal structures within a CSG* is impossible to foresee at design time. When a CES leaves a CSG, the roles of the remaining participants and their operational context may change as well. The same happens when a new vehicle joins the platoon as a platoon participant that later on may take the role of platoon leader. In turn, this leads to a *dynamic change of system borders of a CSG,* which may change the overall functionality of the CSG. For example, a vehicle ahead of the platoon is considered a context object that influences the speed adjustments of the approaching platoon. If the vehicle in front of the platoon decided to join the platoon, then the borders of the initial platoon would be extended.

Using simulation at runtime

Addressing the challenges mentioned above by using simulation may even require using simulation at runtime, which, in turn, puts further requirements on the simulation method.

Meeting real-time deadlines

Firstly, when simulation is used to control the behavior of safety-critical systems, the *real-time deadlines* must be achieved. When system behavior is evaluated at runtime, in a simulated environment, then the simulation must deliver the results on time. This is necessary in order to give the system the chance of executing a safe failover

behavior if the virtual evaluation discovers hazardous behavior of the system under operation.

Secondly, predictive evaluation of system behavior is possible only by achieving efficient simulation models. When system behavior is evaluated at runtime, in a simulated environment, it must execute faster than the wall clock. This imposes a high degree of efficiency on the simulation models that are executed. For example, it may not be feasible to execute detailed simulation models as parts of the interacting platform because this may take too much time. Instead of executing the detailed models, abstractions of the system behavior can be executed. These abstractions must be directed towards the scope of the evaluation. If scheduling behavior needs runtime evaluation in a simulated environment, then the parts of the platform that influence or are influenced by the scheduling will be executed.

Enabling model abstraction to achieve efficiency

However, in order to have accurate evaluation, the efficiency of simulation must balance with the effectiveness of simulation models. In order to perform a trustworthy system evaluation in a simulation environment during runtime, the models must accurately reflect the parts of the system under evaluation. However, because simulation also needs to be efficient, effective simulation can be achieved by using the abstraction models (for efficiency reasons) directed towards the scope of the evaluation. This in turn requires extensive effort during the design time of the system to create accurate models that reflect selected parts (abstraction) of the internal system architecture. For example, to enable evaluation of scheduling at runtime, systems engineers must design the meaningful simulation models of the platform that will be executed during scheduling analysis.

12.3 Simulation Methods

Simulation is a universal solution approach and is based on the application and use of a few basic concepts from numerical mathematics. In our case, simulation models are implemented in software and use numerical algorithms for calculation. We speak of time-discrete, discrete-event, or continuous simulation (continuous time) depending on the mathematical concepts used, which characterize the different handling of time behavior. Simulation tools usually realize a combined strategy. The fact that simulation covers several disciplines, combines different elements of a system, or addresses the system and its context, leads to approaches for a cooperation of different simulations, also called co-simulation. From

www.dbooks.org

a practical point of view, data and result management are important for supporting the simulation activities.

In the area of testing software functions, the three approaches Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), and Hardware-in-the-Loop (HIL) are relevant [VDI 3693 2016]. MIL simulation describes the testing of software algorithms implemented prototypically during the engineering phase. These algorithms are implemented using a *simulation models language*, mostly in the same simulation tool that is also used to simulate the physical system (understood here as the dynamic behavior with its multidisciplinary functions) itself. The SIL simulation describes a subsequent step. The software is realized in the original programming or automation language and is executed on emulated hardware and coupled with a simulation model of the physical system. The third step is a HIL simulation. Here, the program (or automation) code compiled or interpreted and executed on the target hardware is tested against the simulation of the physical system.

Simulation of technical systems usually consists of three steps: model generation (including data collection), the execution of simulation models, and the use of the results for a specific purpose. In the following, we describe the methodology of simulation for these three process steps.

Supporting model creation

In general, the data collection and generation of the models take a lot of effort and time. For virtual commissioning, there are statements that up to two-thirds of the total time is spent on these activities [Meyer et al. 2018]. As a consequence, especially for CESs and CSGs in partially unknown contexts, efficient methods for setting up the model must be provided. Integrating the model generation directly into the development process in order to generate up-to-date models at any time is a good approach, as shown in Chapter 6.

The most common concept for seamless integration of all information relevant in the entire life cycle of a product is product lifecycle management (PLM). It integrates all data, models, processes, and further business information and forms a backbone for companies and their value chains. PLM systems are, therefore, an important source for the creation of simulation models.

With the technical vision of a digital twins approach, the importance of different kinds of models is increased. Digital twins are abstract simulation models of processes within a system fed with real-time data. For more information on supporting the creation of digital twins for CESs, see Chapter 14. Semantic technologies are used to realize the interconnectedness of all information and to guarantee the

openness of the approach to add further artifacts at any time [Rosen et al. 2019]. These semantic connections, frequently realized by knowledge graphs, can be used in future to generate executable simulation models that are up to date with all available information more efficiently.

Enabling model execution

Furthermore, existing models must be combined to form an overall model of different aspects of the system and context. This requires an exchange of models between different tools, which can be solved via co-simulation [Gomes et al. 2017]. The FMI standard [FMI 2019] describes two approaches towards co-simulation. With *model exchange,* only those models that can be solved with one single solver are combined to form an overall mathematical simulation model, whereas *FMI for co-simulation* uses units, consisting of models, solvers, etc. that are orchestrated by a master. On the one hand, this master must match the exchange variables described in the interfaces. On the other hand, it must orchestrate the different time schemes of the different simulators from discrete-event through time-discrete up to continuous simulation [Smirnov et al. 2018]. For efficient simulation of CSGs, the simulation chains must therefore be set up and modified quickly and efficiently as they can change quite often depending on the situation.

In order to set up an integrated development and modeling approach, two aspects must be covered: firstly, different methods must be assembled into an integrated methodology; and secondly, interoperability and integration between different tools must be established in order to set up an integrated tool chain (see Chapter 17). A special focus of co-simulation lies in HIL simulation, which uses real control hardware. The remaining simulation models, with their inherent simulation time, must be executed faster than real-world time to ensure that the results are always available at the synchronization time points with the physical HIL system. Thus, both, the slowest model as well as the orchestration process, must be executed faster than real-world time.

Developing the use of the results

One key goal of simulation is validation and testing of the system behavior. This requires the definition of test cases, the setup of the simulation model, execution of the test cases, and finally, the evaluation of the test. For context-aware CESs and CSGs in particular, this may be a highly complex task with exponentially increasing combinations. Finally, the test results must be compared with the requirements. In Chapter 15, we therefore develop exhaustive testing methods to cope with these challenges.

www.dbooks.org

One way to support the tester is to mark system-relevant information in the requirements and link it to simulation events. A markup language can be used to mark software functions and context conditions within a document. After important text passages in the requirements have been marked, they can be extracted automatically. When the extraction process is completed, the information is linked to the specific signals of the system. This results in a mapping table. Since many simulators, models, and interfaces are used in the simulation of CESs, a central point is created to combine them. In the simulation phase, all signals of the function under test are recorded and stored in log data. These log data contain all signal names and their values for each simulation step. Once the simulation run is complete, the log data can be processed further and linked to the original requirements using the mapping table from the previous phase. This allows the marked text phrases in the requirements to be evaluated and displayed to the user.

Integration into process

Simulation methods are increasingly integrated into the design and development process and used in all phases of the system life cycle [GMA FA 6.11 2020]. Beyond development, validation, and testing, simulation is used during operation with an increasing benefit [Schlegner et al. 2017]. Specific applications include simulations in parallel to operation in order to monitor, predict, and forecast the behavior of the CESs. This means that simulation models must be updated regarding the current state of the systems collaborating in a CSG [Rosen et al. 2019]. Chapter 3 introduces a flexible architecture for the integration of simulation into the systems architecture to support the decision of the system or the operator.

For complex scenarios, the simulation has to cover not only the functional behavior of a single system, but also the combined behavior of the CSG and all relevant aspects, including, for example, the resulting collaboration behavior, the context of the collaborative system, the timing of the systems, and the communication between the systems and with the context. The collaboration functions result from the interaction between the functions of the different systems. All these aspects must be addressed by simulation as early as possible in the design process. It may not be sufficient to test them in a HIL simulation when the implementation of the system has already widely progressed. The MIL and SIL simulations must also address those aspects.

12.4 Application

The methods described above have several applications. First of all, they support development, testing, and virtual integration, especially in early phases of the system design. They also support the development of extended simulation methods such as the ones used for runtime evaluation of system trustworthiness, as presented in Chapter 10; they support the generation of simulation models based on a step-by-step approach, as presented in Chapter 6; and they support the operator during system operation, as presented in Chapter 3. Furthermore, they support system evaluation in real-world scenarios.

Simulation methods for development, testing, and virtual integration

During the design of CESs in particular, simulation methods can help to check the current state of development, verify the correctness and completeness of the current design, and explore the applicability of the next steps and extensions. For collaborative systems, virtual integration of different systems is a special challenge, especially in early and incomplete stages of development. The purpose is to explore the collaborative behavior as early as possible, detect possible hazards and failures when they are much easier to change, and adapt the design of the systems for the solution to these hazards and failures.

Simulating the collaborative behavior in the early stages of development—especially for applications like autonomous driving—should include all relevant aspects of the underlying scenarios, especially context and physical system behavior. Co-simulation approaches can address the challenges involved in such a comprehensive simulation. Chapter 13 provides more details on the possibilities and tools for realizing such simulation approaches.

Simulation methods as a basis for extension

Building trust into collaborative embedded systems requires a sustained evaluation and testing effort that spans from design time to runtime. As detailed in the sections above, simulation is an important technique that enables system and software testing at design time and behavior evaluation during runtime. Within CrESt, as presented in Chapter 10, an extension of existing simulation methods has been realized. These methods either address runtime challenges at design time or enable runtime evaluation of system behavior.

Simulation methods for runtime evaluation

Addressing runtime challenges at design time is enabled by extending the co-simulation method described in this chapter towards integrating the real world (in which collaboration functions and system functions execute on real hardware) with the virtual world (formed by purely virtual entities). This allows the runtime

www.dbooks.org

activation of system functions, for example, to be validated in an extended set of scenarios that are easier and cheaper to explore within a virtual environment.

Building on the challenges and methods described in this chapter, simulation techniques deployable at runtime have been developed. Coupled with monitoring components, simulation can be used for runtime prediction of system behavior emerging from the runtime activation of system functions. When simulation platforms are deployed on CESs, the functional and timing interaction of a collaboration function with system functions and the functional and timing interactions between system functions can be predicted at runtime. For details on how the simulated prediction is performed, see Chapter 10 of this book.

12.5 Conclusion

Simulation methods support the development of CESs, verification and validation of their continuous development, from the conceptual phase when abstract behavioral methods can be coupled through co-simulation and verification of system behavior after detailed models are integrated, up to the final testing of systems before deployment. We have analyzed the benefits and challenges of CESs and of simulation methods that support their development and testing. We have set the basis for future extensions beyond the current state of the art and practice.

In order to realize these technological visions, it is important to consider the economic benefits. This means that the effort and ultimately the cost of deployment must not exceed the benefits. One approach will be a step-by-step realization. This will ensure that advanced simulation methods will be a success factor for validation and testing of CESs.

12.6 Literature

[Alexander and Maiden 2004] I. Alexander, N. Maiden (Eds.): Scenarios, Stories, Use Cases – Through the Systems Development Life-Cycle. Wiley, Chichester, 2004.

[Allmann et al. 2005] C. Allmann, C. Denger, T. Olsson: Analysis of Requirements-Based Test Case Creation Techniques. IESE-Report No. 046.05/E, Version 1.0, June 2005. Fraunhofer-Institute for Experimental Software Engineering IESE, Kaiserslautern, 2005.

[FMI 2019] Functional Mock-up Interface (FMI) Specification 2.0.1, https://fmi-standard.org/, accessed 03/01/2020.

[GMA FA 6.11 2020] R. Rosen, J. Jäkel, M. Barth: VDI Statusreport 2020: Simulation und digitaler Zwilling im Anlagenlebenszyklus. VDI/VDE, 2020 (available in German only).

[Gomes et al. 2017] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe: Co-Simulation: State of the Art. arXiv preprint arXiv:1702.00686, 2017.

[Meyer et al. 2018] T. Meyer, S. Munske, S. Weyer, V. Brandstetter, J. C. Wehrstedt, M. Keinan: Classification of Application Scenarios for a Virtual Commissioning of CPS-Based Production Plants into the Reference Architecture RAMI 4.0. In: Proceedings of the VDI AUTOMATION-18: Seamless Convergence of Automation & IT, 2018.

[Rosen et al. 2019] R. Rosen, J. Fischer, S. Boschert: Next Generation Digital Twin: An Ecosystem for Mechatronic Systems? In: 8th IFAC Symposium on Mechatronic Systems MECHATRONICS 2019: Vienna, Austria, 2019.

[Schegner et al. 2017] L. Schegner, S. Hensel, J. Wehrstedt, R. Rosen, L. Urbas: Architekturentwurf für simulationsbasierte Assistenzsysteme in prozesstechnischen Anlagen. Tagungsband Automation 2017, Baden-Baden 2017 (available in German only).

[Smirnov et al. 2018] D. Smirnov, T. Schenk, J. C. Wehrstedt: Hierarchical Simulation of Production Systems. In: 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE 2018, pp. 875-880.

[VDI 3693 2016] VDI/VDE 3693 Blatt 1:2016-08. Virtual commissioning - Model types and glossary). Berlin: Beuth Verlag.

[Zheng et al. 2014] Y. Zheng, S. E. Li, J. Wang, K. Li: Influence of Information Flow Topology on Closed-Loop Stability of Vehicle Platoon with Rigid Formation. In: 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), IEEE, pp. 2094-2100.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Karsten Albers, INCHRON AG
Benjamin Bolte, itemis AG
Max-Arno Meyer, RWTH Aachen University
Axel Terfloth, itemis AG
Anna Wißdorf, PikeTec

13

Tool Support for Co-Simulation-Based Analysis

The development of collaborative embedded systems (CESs) requires the validation of their runtime behavior during design time. In this context, simulation-based analysis methods play a key role in the development of such systems. Simulations of CESs tend to become complex. One cause is that CESs work in collaborative system groups (CSGs) within a dynamic context., which is why CESs must be simulated as participants of a CSG. Another cause stems from the fact that CES simulations cover various cyber-physical domains. The models incorporated are often managed by different tools that are specialized for specific simulation disciplines and must be jointly executed in a co-simulation. Besides the methodological aspects, the interoperability of models and tools within such a co-simulation is a major challenge. This chapter focusses on the tool integration aspect of enabling co-simulations. It motivates the need for co-simulation for CES development and describes a general tool architecture. The chapter presents the advantages and limitations of adopting existing standards such as FMI and DCP, as well as best practices for integrating simulation tools and models for CESs and CSGs.

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_13

13.1 Introduction

Today's heterogeneous engineering tool environments and the rising number of different systems engineering methods lead to the need for tool interoperability. The development of collaborative embedded systems (CESs) adds another factor to the complexity, as the embedded systems involved must be able to work properly in dynamically changing collaborative system groups (CSGs) and within their environment. This leads to more complex development scenarios, as additional methods must be applied to develop these systems and system groups. In addition, more organizations and stakeholders are involved, each potentially using their own modeling methods and supporting tools. In this context, integrating software development tools is a crucial prerequisite for the efficient engineering of collaborative embedded systems. In order to set up an integrated development and modeling approach, two aspects must be covered: first, different methods must be assembled into an integrated methodology; second, interoperability and integration between different tools must be established in order to set up an integrated tool chain. This chapter focuses on the second aspect. While enabling tool interoperability is important for every kind of CES and CSG development method, this chapter focusses especially on enabling tool interoperability for co-simulation-based analysis methods. Enabling interoperability for these kinds of methods is especially challenging, as it requires data integration not only at the level of model artifacts, but also at the level of a joint execution. The focus of this chapter is complementary to Chapter 12, which covers general simulation-based analysis methods.

After categorizing the different kinds of simulation models and motivating the need for co-simulation, we describe a tool architecture that enables co-simulation, together with the relevant standards FMI and DCP. The concepts and approaches discussed are exemplified by the "*Collaborative Adaptive Cruise Control (CACC)"* vehicle platoon use case (see Chapter 1).

13.2 Interaction of Different Simulations

Simulating CESs and CSGs requires the co-simulation of various highly complex models. There are a large number of models that interact

www.dbooks.org

together to provide the functionality of the system under test (SUT) within its context. These include:

- Environment models (e.g., city with streets and collaborative traffic lights)
- Behavior models (e.g., CACC, platooning control)
- Sensor models (e.g., distance sensor)
- Dynamic models (e.g., vehicle physics)
- Models for the timing behavior of the execution platforms, the implementation, and the communication
- CES/CSG interface models
- Communication models (e.g., wireless car communication)
- Uncertainty models (e.g., sensor and communication uncertainty)

All these models must be interoperable to enable information exchange and time-synchronized execution. Each simulation tool can execute one or multiple of the models listed.

Platooning use case

Let us consider the vehicle platooning use case by way of explanation. A platoon is a collaborative vehicle convoy that uses car-to-car communication based on ad-hoc networks for collaboration. The system that will be used for the platoon control is called Collaborative Adaptive Cruise Control (CACC). This system enables the distance between vehicles to be reduced. The vehicles following the lead vehicle use the data transferred to calculate their relative acceleration.

Analysis

The simulation of complete scenarios can be realized by a co-simulation of different tools and model. While these building blocks apply to CESs and CSGs in general, the concrete types of environment and physics models are typically specific to the use case. To evaluate the behavior of the co-simulation participants, it is important that co-simulation results can be reproduced reliably. In general, there is a set of scenarios that are used repeatedly to compare the results of different co-simulations. Since many embedded systems operate in a safety-critical environment, test scenarios might also be prescribed by safety standards. The scenarios and the focus of the analysis will determine which simulated component parts are necessary to meet the test goals. Only some parts of the functional behavior of the target systems need to be included in the specific co-simulation execution and therefore in the underlying models. Other parts can be either substituted (for example, the pre-processing of sensor data) or omitted entirely if they are not needed for the test execution. The selected level of detail for the different model parts will also depend on the test goals. For parts developed by suppliers, the level of detail

available for the simulation models can also be limited. More abstract models will increase the test performance and allow test and validation in earlier phases of the development process. On the other hand, the significance and the quality of the test results can be limited for abstract models.

Environment simulation

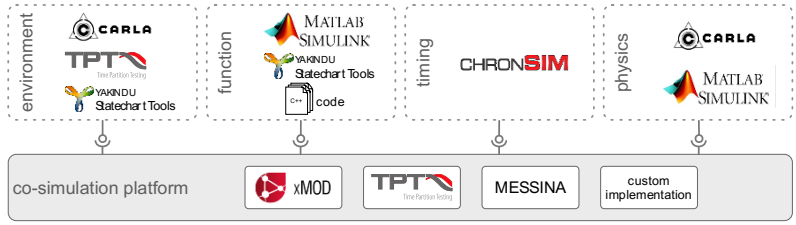


***Fig. 13-1**: Co-simulation model platforms and tools*

The first building block in Figure 13-1 includes tools capable of simulating the context and environment of the systems under test (SUT), such as the roads on which the vehicles are driving and the interfering traffic. For the example use case, the simulation must cover the individual vehicles of the platoon with their specific movements and distances to each other. It should provide input sensor information from the viewpoint of the systems, such as generated camera images or radar vectors; alternatively, depending on the goal of the simulation and the available or desired degree of detail, the simulation should provide pre-processed data such as distances to objects.

An ad-hoc approach is to use simple step-by-step instructions that give the exact sequence of events in simulation scenarios. However, in a co-simulation with numerous simulation models and simulators interacting with each other, this approach is not very well suited to modeling parallel events that might occur. Another common approach is to use statecharts that are more suited for modeling the interaction and collaboration of the models involved (Section 5.2). Tools that support statecharts for test modeling include the YAKINDU Statechart Tools (YSCT) [Yakindu 2020] and Time Partition Testing (TPT) [PikeTec 2020].

Interactive 3D co-simulations support rapid prototyping scenarios for the development of CESs and CSGs. Therefore, the CESs and CSGs are visualized directly within their environment and interactive changes of system behavior models, as well as environment models, are supported in real time. CARLA [Carla 2020] is a vehicle and

www.dbooks.org

environment simulator tailored for evaluating automated driving functions and therefore especially addresses the vehicle platooning use case. It visualizes the environment, the vehicles and their movements, and the complete traffic scenarios based on Unreal Engine [Unreal 2020]. The view of the environment can be captured by multiple sensors attached to different vehicles. The behavior of vehicles can be influenced and set from other simulation tools. CARLA supports interactive changes to objects of the virtual world in real time to create various scenarios and directly visualize the impact on CESs and CSGs in these varying contexts.

Functional simulation

The functional behavior of the CSG can be modeled and simulated with various approaches. MATLAB/Simulink [MathWorks 2020] provides the ability to model and simulate many functions based on sensor, image, or radar data processing. Toolboxes for image processing and autonomous driving functions are available. An exchange with other simulation tools can be provided using co-simulation toolboxes such as the Functional Mock-up Interface (FMI) slave interface. For other scenarios, especially for decision algorithms, modeling the behavior with one or a set of statecharts can be a more suitable approach, and this can be simulated with the YAKINDU Statechart Tools, for example. Another possibility is to include either implemented or generated target code (for example, in C++) in the co-simulation.

Timing simulation

Special simulation tools, such as chronSIM [INCHRON 2020a], augment the co-simulation by incorporating the timing behavior of the software, the execution platforms, the scheduling effects, and the communication between and within systems. In particular, timing effects and delays of the complete event chain, from the sensors, through the processing, to the actuators are derived (see Figure 13-2). The timing simulation replicates the timing of CESs implementing the CSG. Based on the models of the systems and the software, the simulator calculates resulting delays, end-to-end delays for the data processing, as well as potential data losses and more.

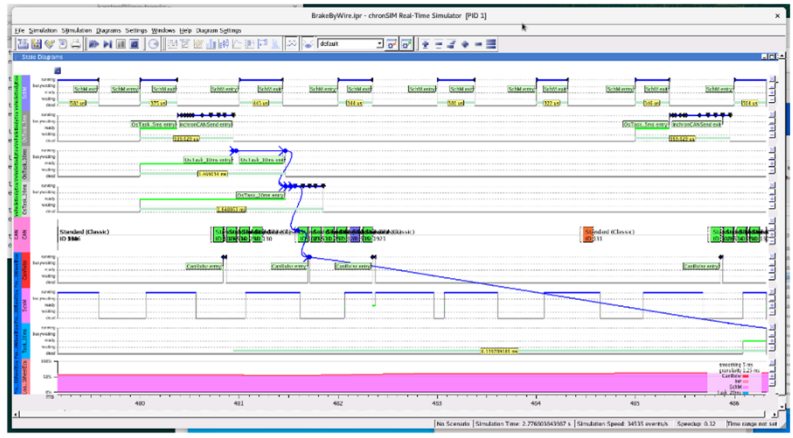


***Fig. 13-2**: An event chain in the timing simulator chronSIM*

Uncertainties resulting from the data propagation, also in wireless communication networks, are therefore incorporated in the overall simulation. Additional uncertainties are part of other models and can arise from inaccurate sensor measurement and processing of sensor data, which should be reflected by the overall simulation. Fault tolerance of the system under test (SUT) with respect to context changes can be considered with predefined configuration parameters for the tool platform, such as typical uncertainty distributions. Therefore, multiple varying configurations could be derived (semi-automatically) from rule sets or automata [PikeTec 2020b] defined in test and scenario models.

Physical dynamic simulation

Another relevant aspect is the physical, dynamic model of CESs. For the vehicle platooning and autonomous transport robots use cases, for example, the speed reduction by braking under various conditions, the steering capabilities for trajectory planning, and so on are part of these models. There are multiple solutions available with various levels of complexity and accuracy. Again, MATLAB/Simulink provides solutions and CARLA also includes a simplified dynamic model.

Co-simulation platforms

The simulation of the physics and environment, together with other co-simulation participants, must be executed in a time-synchronized fashion. Usually, 3D visualization and physics engines have their own timing and try to update the environment for rendering new images as quickly as possible to provide a real-time visualization. The joint execution of the simulation models and tools involved requires a co-simulation platform. Examples are MESSINA

www.dbooks.org

provided by Expleo, xMOD provided by FEV, and TPT provided by PikeTec. Custom implementations can also add co-simulation platform features to tools like CARLA.

13.3 General Tool Architecture

This section provides details of a proper tool architecture, co-simulation standards, and their application to the CES and CSG simulation.

Master-slave architecture

The need for a time-synchronized execution naturally leads to a master-slave architecture. This architecture defines two roles for tools participating in the co-simulation: the *co-simulation master* and the *co-simulation slave*. The *master* manages a set of *slaves*, coordinates interaction between them, handles time synchronization, and makes co-simulation results accessible for subsequent analysis steps. The *co-simulation slave* provides a simulation API (application programming interface), which is used by the co-simulation master to proceed with the simulation, together with a description of the slave's functional interface, such as signals that are consumed or produced by other slaves.

Co-simulation configuration

This description of slave interfaces forms the basis for the configuration of a *co-simulation*. Engineers have to specify the mapping between the interfaces of the slaves involved to realize the intended data flow between models. In addition, the simulation duration and time step for updating the simulation models are defined according to the requirements of the co-simulation participants involved. This is necessary to achieve the required accuracy during the simulation.

This configuration can be defined using a *co-simulation configuration service,* which can be either a tool with a UI or an automated service that applies a transformation from an existing system or CSG model that contains the required information. In any case, it is the co-simulation master's obligation to process this configuration correctly.

Distributed co-simulation

As the co-simulation of many co-simulation components can be resource-intensive and time-consuming, especially when using real-time 3D rendering, parallelization of the co-simulation components over multiple processing units is beneficial to provide more computing resources and to decouple components such as the co-simulation master, modeling tools, and the visualization. This does not

mean that the tools must be used in a distributed computing context; they can also run on one computer without any distribution.

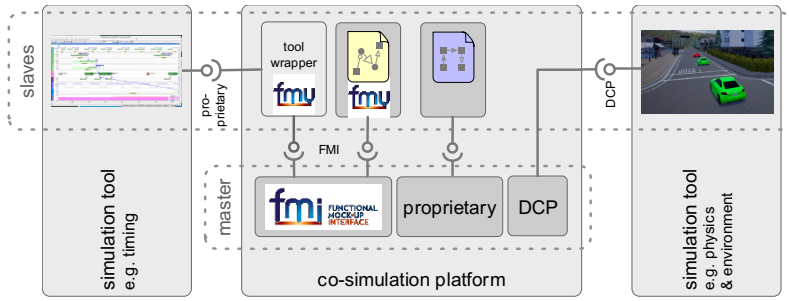


***Fig. 13-3**: Co-simulation tool architecture*

Tool interoperability

Figure 13-3 shows the exemplary architecture of an interactive co-simulation. To enable a tool platform to support co-simulation, various tools and models must be made interoperable. We identified two standards as particularly relevant in the context of developing CESs and CSGs. The first is *Functional Mock-up Interface (FMI)* and the second is the *Distributed Co-Simulation Protocol (DCP)*. These standards can be applied by tool developers as a basis for setting up co-simulation features or in combination with existing proprietary solutions to extend tool interoperability. Both standards comply with the main architectural principles and will be introduced in subsequent sections.

13.4 Implementing Interoperability for Co-Simulation

FMI standard

The FMI for co-simulation standard [Modelica Association 2019a] addresses the integration of heterogeneous simulation models and tools that match the existing constraints for the development of CESs and CSGs. It defines the required technical master-slave interface for a master-slave architecture.

Each model is provided by a co-simulation slave called a Functional Mock-up Unit (FMU). An FMU is a zip file containing at least one executable binary library, along with an XML file that includes the interface definition for the slaves. Libraries for multiple platforms can be included to support portability. The FMI co-simulation master dynamically loads and executes the binary libraries of all slaves. The master-slave interface itself is defined as an API in the C programming

www.dbooks.org

language, with an underlying state machine that defines the order of interface calls.

The FMU interface concept is based on a data-flow paradigm. A model defines a set of input and output variables with simple data types, such as real, integer, and string. The FMI master proceeds with the simulation step by step. In each step, all FMUs are provided with the current input values and are then executed. Finally, the output variables are propagated to the input values for the next execution cycle. Thus, data exchange occurs only between successive execution steps.

FMI limitations

FMI brings with it some constraints that may be relevant when considering the FMI standard compared to proprietary approaches. *Structured data types* do not have a direct counterpart in FMI but must be substituted by simple variables, which flattens the hierarchical data structure. *Events* can only be mapped to changes of input or output values, such as rising and falling edges, which requires the application of conventions between all co-simulation participants. The same is true for synchronous *operation* calls, which would require a complex protocol consisting of call and return events. Finally, *behavioral types* supporting dynamic reconfiguration in CSGs cannot be mapped in a meaningful way.

Time synchronization

The co-simulation master controls the simulation progress and is thus responsible for the time synchronization between all co-simulation components involved. In FMI, each FMU implements the *fmi2DoStep* function which gets the current simulation time point and the duration of the next time step as parameters. The FMI master decides on the step size, which can be of fixed or variable length. The slaves proceed with the simulation for the requested step size. Slaves can use a virtual clock to provide faster than real-time executions, which is relevant for long-running simulations or repeated test scenarios.

Co-simulation participants, such as visualization and physics engines, might provide their own timing behavior that must be synchronized. The first possibility is to use an external co-simulation master to set the timing. Therefore, slaves (e.g., CARLA) must provide a time synchronization interface. Second, if only one additional tool with its own timing is used, this can be extended by a custom implementation of a co-simulation master.

Co-simulation slave generator

To cope with the standard, for each participating simulation model, a *co-simulation slave generator* transforms the model into a standalone executable co-simulation slave. The code generation typically involves the generation of a code layer that implements the

required functionality for data exchange and time synchronization according to the simulation API realization used (e.g., FMU Interface) so that co-simulation-slaves act as a black-box component for the co-simulation master. Code generation can be used to support different software and embedded) hardware platforms to reduce manual implementation efforts and to improve quality. Figure 13-4 shows how the co-simulation slave generation concept is applied for creating FMUs.

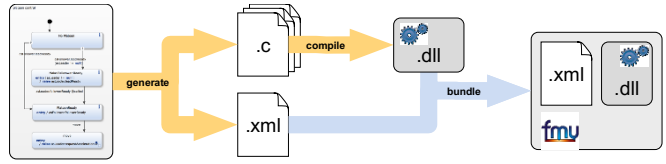


***Fig. 13-4**: Building FMUs using C-code generation*

The generator derives the meta-information description from the model's interface definition to describe input and output variables. In addition, the code for the model execution library is generated, which consists of code to access or implement the executable model and an adapter for the FMU simulation API. If a modeling tool already provides a code generation or interpreter for its models, then it is good practice to reuse these and just add a generator for the required adapter code. Finally, the model description and executable library are bundled as an FMU zip file.

13.5 Distributed Co-Simulation

Distributing models in a co-simulation across multiple platforms is another key interest for realizing complex simulations for CESs and CSGs. A communication infrastructure for connecting the different co-simulation participants is required. While distribution is a proposed FMI use case, the realization of distribution and the communication layer are left untouched by the standard.

Co-simulation tool wrapper

If an external tool is required for simulating a slave model, a direct execution within the *FMI master* is not possible as it assumes the complete execution of the co-simulation within a single multi-threaded process. A *co-simulation tool wrapper* is a specific *co-simulation slave* that, instead of executing the model itself, delegates all execution requests to the external simulation tool. This requires a communication layer that must handle data exchange, time synchronization, and invocation of simulation. Additionally, the co-

www.dbooks.org

simulation slave must adapt its simulation API to the communication layer and handle data binding. The standard does not prescribe the concrete communication protocol and therefore any existing protocol can be reused.

Distributed Co-Simulation Protocol - DCP

To overcome the need for a tool-specific wrapper, the Distributed Co-Simulation Protocol (DCP) was developed as an open, accompanying standard to FMI [Krammer et al. 2018]. It standardizes the distribution of models on different software and hardware platforms, which is particularly important for handling models bound to specific execution environments and to increase simulation performance. DCP focusses particularly on the following aspects. First, DCP is specifically designed for the integration of real-time and non-real-time systems simultaneously. Therefore, it is possible to perform co-simulations that combine hardware setups with digital models. Second, DCP can be combined with other standards, such as FMI and proprietary solutions. Third, DCP supports a wide range of communication protocols (UDP, TCP, CAN, USB, Bluetooth 3) to ensure interoperability on the application layer regardless of the communication medium [Modelica Association 2019b].

DCP also applies a master-slave architecture. As in FMI, DCP requires XML-based configuration data and defines a state machine for the execution and communication life cycles of slaves. The communication to the master and other slaves is handled via protocol data units (PDU) defined by DCP. Thus, the specification provides a precise basis and guidance for the implementation of DCP on the master and slave side by tool developers.

The option to flexibly distribute master and slave to different hardware and operating systems can be used effectively in CES and CSG co-simulations to increase overall simulation performance and to integrate interactive simulations or concrete hardware. In particular, DCP enables synchronization in either real time or non-real time with other non-DCP co-simulation slaves, such as plain FMUs or a co-simulation tool wrapper using proprietary protocols.

DCP limitations

The initial implementation effort for DCP is definitely higher compared to reusing an existing proprietary protocol. However, once implemented, masters and slaves can interoperate natively with other platforms that support the standard. With regard to modeling concepts, DCP has the same limitations as FMI.

13.6 Analysis of Simulation Results

During and after the execution of the simulations, a co-simulation analysis service is necessary to evaluate and extract the simulation results for conclusions and follow-up decisions.

For certain scenarios, the success can be determined by checking whether the co-simulation participants reach/avoid a failure state or meet predefined goals. The different simulators and tools can track such conditions directly during the simulation execution.

For evaluations that cannot be executed directly in the co-simulation platform, a useful approach is to record the execution and system states in one or a set of trace files during the simulation. Based on the information thus gathered, the fulfillment of requirements can be checked and statistical information on the behavior can be derived.

Analysis and reporting tools read the machine-readable simulation traces for further processing. The use of open formats can help with processing of simulation traces from a larger set of tools in a co-simulation. The information available from the different simulations will be quite different, which will affect the required trace formats. For example, state transitions for the state machines can be recorded in a behavior model, allowing the engineers, for instance, to validate models on an even deeper layer and enable gray and white-box verification. For the timing behavior, execution states, events, and data processing chains should be recorded. A synchronization of the trace files from the different simulation tools is necessary for the evaluation of cross-over aspects. Time stamps from the common time base or shared frequent events can make synchronization easy.

Another advantage of traces is that they make it easier to determine the reasons for an observed behavior, such as a failed requirement. Critical situations can be visualized and explored by tools like chronVIEW [INCHRON 2020b].

13.7 Conclusion

In this chapter, we addressed the task of enabling tool interoperability for co-simulation-based analysis methods for CESs and CSGs. A particularly challenging aspect for enabling tool support for co-simulations is that the tool integration must facilitate a joint execution of model artifacts that are integrated at a data level.

A distributed master-slave architecture with well-defined interfaces is the basis for orchestrating and coordinating heterogeneous models and tools into a co-simulation. The FMI and

www.dbooks.org

DCP standards support this architecture. FMI-compliant models can be reused and executed on different co-simulation platforms, which may serve a specific purpose. DCP-enabled platforms and tools can easily be connected. Both standards can be combined with existing proprietary solutions, enabling reuse of simulation tools, platforms, and communication infrastructure.

The data-flow-oriented approach of FMI and DCP has limitations with regard to applicable modeling concepts and this constrains the applicability for co-simulation scenarios that require dynamic reconfiguration of CSGs. Here, proprietary approaches may be a better fit. The standards also do not define a model for connecting slaves. This is the responsibility of the concrete master implementations. Thus, CSG models that describe such model relationships must be mapped specifically for each master implementation and are not easy to reuse.

The distributed setup enables integration of heterogeneous co-simulation tools, which may even support interactive changes to the models during runtime. As a result, the development process can potentially be improved in certain ways. First, an explorative development of models without time-consuming code generation steps is provided. Second, many functional components from various vendors can be combined for rapid prototyping and early testing scenarios. Third, the visualization of test scenarios has potential to improve the communication with the stakeholders involved across various organizations.

Co-simulation improves verification and validation of CESs and CSGs. Trace information from all co-simulation participants enables required analysis methods and tools to enhance verification and allow a statistically rich evaluation. Simulation tools contribute environment, function, timing, uncertainty, and physical models in a scope and a level of detail that is appropriate for different scenarios to the co-simulation. The resulting co-simulation thus better reflects real-world scenarios, which improves the generalizability of the validation and verification results.

13.8 Literature

[Carla 2020] CARLA – Open Source Simulator for Autonomous Driving Research: *https://carla.org*. accessed on 05/08/2020.

[Expleo 2020] MESSINA – Test Automation and Virtual Validation for Embedded Systems: https://www.expleo-germany.com/en/produkte/messina/, accessed on 05/08/2020.

[INCHRON 2020a] chronSIM – Model-Based Simulation of Embedded Real-Time Systems: https://www.inchron.com/tool-suite/chronsim, accessed on 05/08/2020.

[INCHRON 2020b] chronVIEW https://www.inchron.com/tool-suite/chronview/, accessed on 05/08/2020.

[Krammer et al. 2018] M. Krammer, M. Benedikt, T. Blochwitz, K. Alekeish, N. Amringer, C. Kater, S. Materne, R. Ruvalcaba, K. Schuch, J. Zehetner, M. Damm-Norwig, V. Schreiber, N. Nagarajan, I. Corral, T. Sparber, S. Klein, J. Andert: The Distributed Co-Simulation Protocol for the Integration of Real-Time Systems and Simulation Environments. In: Proceedings of the 50th Computer Simulation Conference, 2018.

[MathWorks 2020] Simulink – Simulation and Model-Based Design https://www.mathworks.com/products/simulink.html, accessed on 05/08/2020.

[Modelica Association 2019a] Modelica Association Standard: FMI – Functional Mock-Up Interface Specification Document 2.0.1, 2019.

[Modelica Association 2019b] Modelica Association Standard: Distributed Co-Simulation Protocol (DCP). Specification Document 1.0.0, 2019.

[PikeTec 2020] TPT: Control Testing Made Easy https://piketec.com/tpt/, accessed on 05/08/2020.

[PikeTec 2020b] Automatic Test Case Generation in TPT https://piketec.com/tpt/testcase-generation/, accessed on 05/08/2020.

[Unreal 2020] UNREAL ENGINE https://www.unrealengine.com, accessed on 05/08/2020.

[Yakindu 2020] YAKINDU Statechart Tools https://www.itemis.com/en/yakindu/state-machine/, accessed on 05/08/2020.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Emilia Cioroaica, Fraunhofer IESE
Thomas Kuhn, Fraunhofer IESE
Dimitar Dimitrov, Siemens AG

14

Supporting the Creation of Digital Twins for CESs

One important behavioral aspect of collaborative embedded systems (CESs) is their trustworthiness, which can be assessed at runtime by evaluating their software and system components virtually. The key idea behind trust evaluation at runtime is the assessment of system interactions and consideration of an extended set of actors that influence the dynamicity of these systems. In this sense, the behavior of collaborative embedded systems and collaborative system groups (CSGs) is part of a more complex behavior of digital ecosystems that form around the collaborating systems. One way of performing runtime virtual evaluation of such complex behavior is through the implementation of digital twins (DTs). DTs are executable models fed with real-time data that allow behavior to be observed and analyzed in concrete technical situations. The use of digital twins enables goals to be evaluated in holistic scenarios at three different levels: strategic level, tactical level, and operational level, as we present in this chapter.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_14

14.1 Introduction

By considering the actors that interact directly and indirectly with collaborative embedded systems (CESs), the concept of collaborative embedded systems and collaborative system groups (CSGs) extends towards the notion of *digital ecosystems*. Within an ecosystem, actors such as organizations, developers, and users have a multitude of goals, and may act not only in cooperation but also in competition. These dynamics influence the behavior of CESs within CSGs directly and indirectly.

Collaborative systems are part of complex ecosystems

In [Cioroaica et al. 2019], we have defined trust-based digital ecosystems where the trustworthiness of a collaborator is computed rather than being granted by default. In the assessment of a digital ecosystem from the trust perspective, a *trustor* is the user of a service who can trust a *trustee,* who is the provider of the service, to satisfy its needs and expectations linked to a *trustum,* which is the service provided. Consider an example at the level of collaborating systems in the automotive domain: a following vehicle (trustor) uses the coordination commands (trustum) to adapt the speed of a lead vehicle in a platoon (trustee). Similarly, a vehicle that intends to join a platoon (trustor) uses the goal information communicated (communication service is the trustum) by the platoon leader (trustee) to make its decision. The architectural model presented in this chapter supports the creation of digital twins for holistic trust evaluation.

Trust results from reputation computed in multiple verification scenarios. From a safety perspective, the reputation of the leading vehicle must be evaluated to ensure trust in the ecosystem that is built around the platoon. In the model that we introduce in this paper, the quality of service (QoS) provided by a product has an impact on the health of the ecosystem. According to [da Silva et al. 2017], the health of an ecosystem is linked to how well the business develops. For example, wrong or delayed commands lead to string instability within a platoon. String instability is characterized by sudden braking and acceleration, which in turn create an increase in fuel consumption instead of a reduction (business goal). This impact is analyzed by providing a structural hierarchy of the relationships between the quality of service and the business goals of the actors. The computation of trust in a collaborator starts with the evaluation of the *operational goals* of the system. The results are used to evaluate the *strategic goals* of the ecosystem that can be achieved by CESs. If we

www.dbooks.org

return to the context of forming a vehicle platoon, where a system function sends context information that is inaccurate or even intentionally wrong, then the *tactical goal* of the CESs to form an effective vehicle platoon will not be achieved. This has an impact on the *strategic goal* of reducing fuel consumption, with direct impact for the participants in a CSG. However, if this type of behavior is discovered early enough, the vehicle providing the malicious service will not be granted access to the ecosystem. Therefore, successful evaluation of strategic goals relies on proper evaluation of the tactical goals, which in turn relies on the evaluation of operational goals for every system engaged in a collaboration. The hierarchical nature of decision-making based on the main differences and distinctions between three types of decisions—namely strategic decisions, tactical decisions, and operational decisions—is described in [Hollnagel et al. 2003] and [Molen et al. 1988]. In our reference architecture, we use a similar hierarchy to structure the goals within an ecosystem by considering systems, system components, and actors.

14.2 Building Trust through Digital Twin Evaluation

Fulfilment of strategic goals of a collaboration relies on the fulfilment of tactical goals which, in turn, relies on the correct implementation of operational goals

Given the distributed provision of hardware resources and software components, the formation of collaborative systems through runtime activation of system functions requires a runtime evaluation of the hardware–software interaction as well. For this particular situation, [Seaborn and Dullien 2015] have shown that specific hardware–software interaction patterns may be faulty and may lead to serious system failures that manifest into security threats. This would be disastrous for CESs and implicitly for the health of the ecosystem formed around the CSG. A runtime assessment and evaluation of the level of trust in the components of a CES is therefore required.

A novel approach to building trust in a software component without executing its behavior in real operation is by evaluating its digital twin at runtime. We have introduced such an approach in [Cioroaica et al. 2019]. In the early days of autonomous computing systems, reputation was seen as a good indicator of the level of trust in a system. The authors of [Kephart et al. 2003] propose storing information about a system's reputation in order to address the need to compute the trustworthiness of potential collaborators. The notion of a digital twin (DT) was initially introduced by NASA [Shafto et al. 2012] as a realistic digital representation of a flying object used in laboratory testing activities. Since then, the notion of DT has also been

adopted in the emerging Industry 4.0 [Rosen et al. 2015] to represent the status of production devices and to enable the forecast of the impacts of change. The reference architecture presented in this chapter enables the creation of digital twins for the whole ecosystem. The digital twin provides a machine-readable representation of the goals of the entities that are part of the ecosystem and supports their trust evaluation through execution of verification scenarios that reflect their dynamic behavior.

Digital twins provide a machine-readable representation of goals

Figure 14-1 depicts an example of a basic classification of system goal types in the supertype-subtype hierarchy. The goals depicted in boxes with a dashed line represent default types that can be reused in any domain. The goals depicted in boxes with a continuous line represent extensions for a specific domain. This classification is supported by evidence showing that, besides its declared well-intended contributions to a collaboration, a system can also have contributions with malicious intent. These intentions can be exposed through malicious behavior caused by malicious faults [Avizienis et al. 2004]. The malicious behavior of a system represents the undeclared competing goals of actors introducing systems and system components on the market.

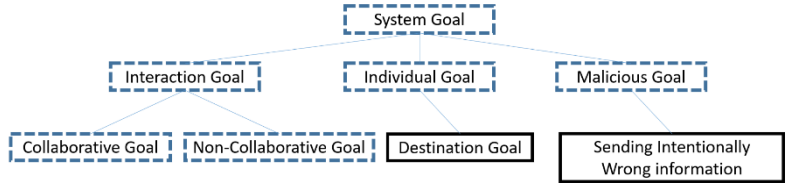


Fig. 14-1: *Classification of Goal Types*

The creation of digital twins of the ecosystem and ecosystem participants is enabled by an architecture that contains a description of goals and provides support for the reputation computation in specific verification scenarios. The scenarios describe concrete technical situations in which decisions need to be taken — for example, joining or leaving a platoon. The digital twin of an ecosystem enables information access at runtime and supports a CES in making the decision of whether or not to join a specific platoon. In Figure 14-2, we depict the ecosystem perspective on CESs. CESs and CSGs exist within digital ecosystems. In literature, there are two types of digital ecosystems: software ecosystems, formed around software products [Manikas et al. 2013], and smart ecosystems, formed around cyber-physical systems, such as automotive smart ecosystems [Cioroaica et al. 2018]. Within an ecosystem, actors can play different roles, such as

www.dbooks.org

manufacturer, distributer, user, subcontractor, etc. and can have a multitude of goals of various types — for example, collaboration, competition, increase in revenue, etc. The system behavior is the asset that enables goal satisfaction.

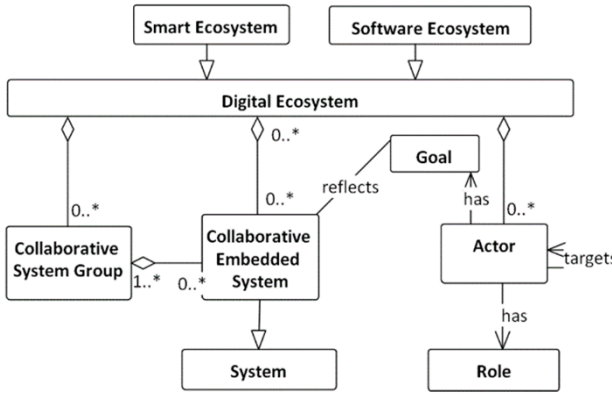


Fig. 14-2: *Ecosystem Perspective on CES*

Figure 14-3 and Figure 14-4 show the instantiation of the architecture that enables the creation of digital twins in the automotive and smart grids domains. Given its context-specific operational capacity, an embedded system by itself is meant to operate to achieve dedicated business goals.

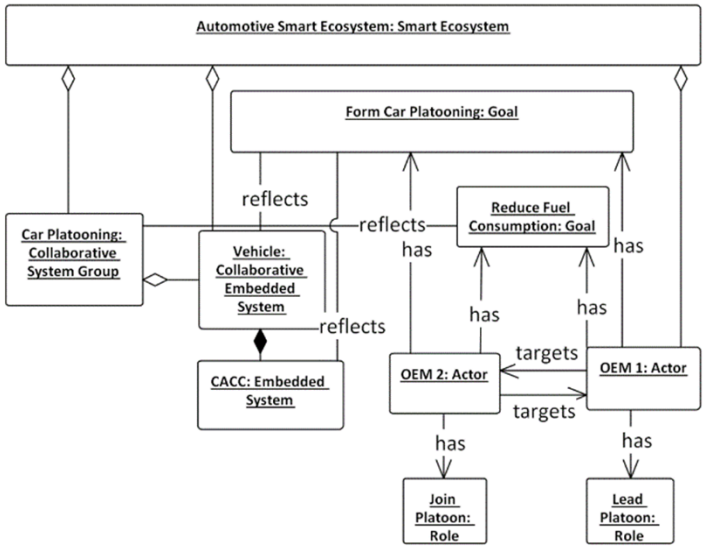


Fig. 14-3: *Example instantiation in the automotive domain*

However, through communication and collaboration with other embedded systems, enhanced functionality can be achieved. Depending on the goal and the role an actor has in an ecosystem, other actors are targeted. The operative part of an ecosystem is formed by the systems that collaborate with each other at runtime in order to fulfill enhanced business goals of different organizations or the same organization. Communication and collaboration are realized through an exchange of data and functions and can be between embedded systems located in the same system or embedded systems located in different systems. In the first case, communication is realized through dedicated communication buses; in the latter case, the communication between embedded systems is realized through Internet communication.

The collaborative goals of systems are influenced by business goals, which in turn depend on risks, such as economic risks. The concepts of risks and goals are related to actors that play certain roles in a collaboration. In Figure 14-5, examples of roles are the user and the provider. A holistic evaluation of embedded systems that collaborate in the field must take all these aspects into account. Figure 14-6 presents an instantiation in the automotive domain.

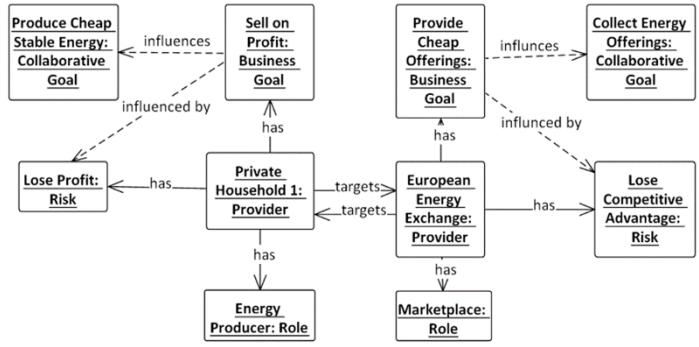


Fig. 14-4: *Example instantiation in the smart grid domain*

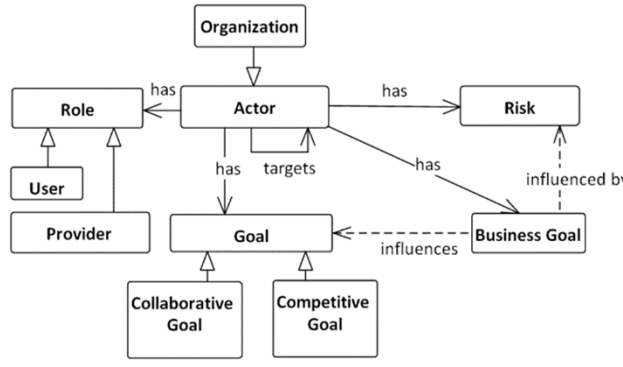


Fig. 14-5: *Business perspective on Collaborative Systems*

www.dbooks.org

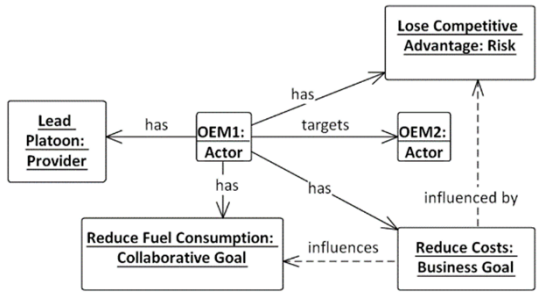


Fig. 14-6: *Instantiation of the business view in the automotive domain*

Figure 14-7 depicts the evaluation of trust through computation of reputation. From the point of view of a collaborator, a system is a resource with functional behavior and non-functional properties. Through its behavior or through the service it is providing, the resource influences the reputation of an Original equipment Manufacturer (OEM) that introduces the system on the market. A reputation of a component is a combination of the initial reputation of an OEM, calculated when the system is first introduced on the market, and the runtime reputation computed via a series of algorithms. The computation of runtime reputation is linked to verification scenarios that describe the context in which the resources are evaluated. Verification scenarios are linked to functional and non-functional requirements that reflect the expectations of the user for the system behavior.

Requirements are provided by the users of services. Based on requirements, verification scenarios are defined in order to evaluate the reputation of resources. The resources provided reflect the goals of the actors during collaboration with other actors. This kind of verification scenario can, for example, evaluate individual goals of vehicles wanting to join platoons for compatibility. Only the vehicles that have compatible routes are granted access to the platoon, and implicitly to the ecosystem. Other verification scenarios can evaluate the expectations with regard to the exchange of services. If, for example, a vehicle requests exchange of information every 100 ms, it should avoid joining a group of vehicles that exchange information every 100 ms. If the internal system functions of a vehicle are activated and checked every 200 ms, joining a platoon that requests information exchange every 100 ms may cause synchronization issues.

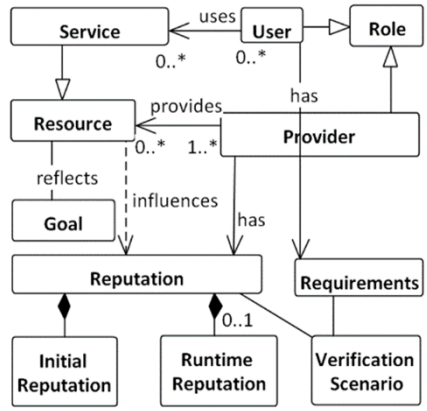


Fig. 14-7: *Computation of trust based on runtime verification scenarios*

14.2.1 Demonstration

In this section, we present scenarios from the automotive and smart grid domains that benefit from the instantiation of digital twins of their ecosystems based on the reference architecture introduced in the previous section.

Automotive Smart Ecosystems

At the entry point for a highway, consider a scenario in which a vehicle (CES) activates a collaboration function (SW component) and corresponding running specifications which are digital twins. The collaboration function enables the vehicle to join or form vehicle platoons (CSG). If the vehicle starts forming a vehicle platoon, it becomes the leader of that platoon (it has the role type “Platoon Leader”). The other vehicles with the same goal (collaborative goal) can be members of the same platoon (assigned the role type “Member”). Besides having the same collaborative goal, the vehicles must have fitting individual goals in order to join beneficial collaborations. For example, only vehicles with the same collaborative goal of being part of vehicle platoons and moving towards similar destinations (Reaching Destination Goal as a subtype of Individual Goal) may be part of the same platoon. When another vehicle approaches an existing platoon, it requests the digital twin of the ecosystem containing the platoon and checks whether its goals fit the goals within the ecosystem. If, for example, the vehicle approaching the platoon has the goal of reaching a destination that is not compatible with the route of the platoon, then it will not join this particular platoon. The collaboration function part of the ecosystem

www.dbooks.org

operates on an ECU (embedded control unit, which is an embedded system). It reads context information such as speed and distance communicated by the vehicle in front. According to our architecture, the process of information reading is an operational goal, which is enabled by the *context information reading* service. The collaboration function sends this information to system functions. The process of sending the information is a *data transfer* service. The system functions are responsible for maintaining the maximum distance between vehicles in a platoon while maintaining the minimum safe distance. According to our architecture, the process of managing the distance is a service associated with the smart agent, via a service assignment with the role type "Provider." The service has a contract of the contract type "Specification." The maximum distance is the distance that allows the platoon members to benefit from reduced air friction and implicit reduction of fuel consumption (strategic goal).

If the information provided by a system function is wrong—if, for example, the vehicle in front transmits that the distance is 7 m, but the actual distance is 5 m—then the system function might accelerate. According to our architecture, the acceleration is an operational goal. The vehicle can accelerate until it learns from its own sensors that the minimum safety distance has been violated, and then it will brake immediately. Acceleration followed by instant braking creates string instability in the platoon and implicit higher fuel consumption. In the worst case, this could cause a crash. By using a digital twin of the digital ecosystem instantiated with our reference architecture, a violation event will be recognized before it actually happens. Specifically, the reputation score of the vehicle causing a violation and its associated actors will become negative (based on the output of the reputation computation that compares the observations of the distance properties with the contract). As a result, the vehicle will not be granted access to this ecosystem.

By capturing the system decomposition, our reference architecture forms the basis for instantiating a digital twin of the ecosystem. This allows the identification of failure cases at the system level, thus supporting the replacement of faulty or malicious components. A system function that does not perform according to its specifications can be replaced with an improved version of itself or another system function provided by another organization that is an actor in the ecosystem. A digital ecosystem has specific sets of verification scenarios that compute the reputation of its participants. If the requirements and expectations of the verification scenarios and

of a vehicle that wants to join the ecosystem are compatible, the vehicle is granted access to the ecosystem and it can decide to join it.

Smart Grids

In a smart grid, power can be generated by a large variety of decentralized energy resources (DERs), such as wind turbines or photovoltaic plants, each providing a small fraction of the energy. By integrating a connector box (CES) on a DER, the DER is capable of joining (tactical goal) a virtual power plant (VPP) (digital ecosystem) to sell the energy produced (VPP associated with the business goal). Through the deployment of collaboration functions, connector boxes can become fully autonomous and form coalitions (CSGs) in order to provide flexible quantities of energy (strategic goal) when requested by a distributed system operator (actor assigned by actor assignment to the CSG with the role type "Customer"). When no flexibility of energy production is achieved, sanctions are applied in the form of shutdown of the DER (risk associated with the strategic goal of providing a flexible quantity of energy). Therefore, when a member of a coalition cannot fulfill its commitment, a replacement must be found (tactical goal). In order to find the right replacement, the connector boxes must communicate accurate information about their state (operational goal enabled by broadcasting information regarding its status service). The connector boxes must send their status at least once every 15 minutes (specification of the property of the "status broadcast frequency" property type in the contract of the "status broadcast" service). For example, if one connector box does not communicate its status or does not communicate its status correctly, a broadcast for bids cannot start and the flexibility for providing energy will not be achieved.

When a smart agent inside a connector box wants to take part in a collaboration, it must compute the level of trust in the ecosystem that forms around the collaborating systems. This can be achieved by querying the digital twin of the ecosystem, which provides information about the goals of different DERs together with their behavior evaluation in various verification scenarios and their associated reputations.

14.3 Conclusion

In this chapter, we presented a reference architecture that enables automatic computation of trust in ecosystems and ecosystem

www.dbooks.org

components. The reference architecture captures the main concepts and relationships within an ecosystem and can be used to instantiate digital twins. The reference architecture was developed to be flexible and be customizable in various application domains. We showed the expressiveness and reusability of the architecture by providing examples of its instantiation in scenarios from both the automotive and energy domains. Currently, the reference architecture provides the high-level logical view of ecosystems. In future work, we aim to extend the reference architecture with additional views such as the following: a use case view to capture the key usage scenarios; an interaction view to explicitly model the processes and interactions within the domain; and a deployment view to capture the implementation decisions for systems based on the reference architecture. Additionally, because trust evaluation requires detailed analysis of goals, ongoing work is directed towards detailing the goal classification for trust computation.

14.4 Literature

[Avizienis et al. 2004] A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing. In: IEEE transactions on dependable and secure computing, Vol.1, 2004, pp. 11–33.

[Cioroaica et al. 2018] E. Cioroaica, T. Kuhn, T. Bauer: Prototyping Automotive Smart Ecosystems. In: 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2018, pp 255–262.

[Cioroaica et al. 2019] E. Cioroaica, S. Chren, B. Buhnova, T. Kuhn, D. Dimitrov: Towards Creation of a Reference Architecture for Trust-Based Digital Ecosystems. In: Proceedings of the 13th European Conference on Software Architecture - Volume 2. ACM, 2019, pp. 273–276.

[Cioroaica et al. 2019] E. Cioroaica, T. Kuhn, B. Buhnova: (Do Not) Trust in Ecosystems, In: Proceedings of the 41st International Conference on Software Engineering, 2019.

[da Silva et al. 2017] S. da Silva Amorim, F. S. S. Neto, J. D. McGregor, E. S. de Almeida, C. von Flach G Chavez: How Has the Health of Software Ecosystems Been Evaluated?: A Systematic Review. In: Proceedings of the 31st Brazilian Symposium on Software Engineering. ACM, 2017, pp. 14–23.

[Hollnagel et al. 2003] E. Hollnagel, A. N˚abo, I. V. Lau: A Systemic Model for Driver-in-Control, 2003.

[Kephart et al. 2003] J.O. Kephart, D.M. Chess: The Vision of Autonomic Computing. In: Computer 2003, pp 41–50.

[Manikas et al. 2013] K. Manikas, K. M. Hansen: Software Ecosystems – A Systematic Literature Review. In: Journal of Systems and Software Vol. 5, 2013, pp: 1294–1306.

[Molen et al. 1988] H. H. Van der Molen, A. M. Bötticher: A Hierarchical Risk Model for Traffic Participants. In: Ergonomics, vol. 31, no. 4, 1988, pp. 537–555.

[Rosen et al. 2015] R. Rosen, G. von Wichert, G. Lo, K. D. Bettenhausen: About the Importance of Autonomy and Digital Twins for the Future of Manufacturing. In: IFAC-PapersOnLine48, Vol.3, 2015, pp. 567–572.

[Seaborn and Dullien, 2015] M. Seaborn, T. Dullien: Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges, Black Hat15, 2015.

[Shafto et al. 2012] M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne, L. Wang: Modeling, Simulation, Information Technology and Processing Roadmap. In: National Aeronautics and Space Administration, 2012.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Ilias Gerostathopoulos, Technical University of Munich
Alexander auf der Straße, University of Duisburg-Essen

15

Online Experiment-Driven Learning and Adaptation

This chapter presents an approach for the online optimization of collaborative embedded systems (CESs) and collaborative system groups (CSGs). Such systems have to adapt and optimize their behavior at runtime to increase their utilities and respond to runtime situations. We propose to model such systems as black boxes of their essential input parameters and outputs, and search efficiently in the space of input parameters for values that optimize (maximize or minimize) the system's outputs. Our optimization approach consists of three phases and combines online (Bayesian) optimization with statistical guarantees stemming from the use of statistical methods such as factorial ANOVA, binomial testing, and t-tests in different phases. We have applied our approach in a smart cars testbed with the goal of optimizing the routing of cars by tuning the configuration of their parametric router at runtime.

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_15

15.1 Introduction

The behavior of CESs and CSGs is difficult to completely model a priori

Collaborative embedded systems (CESs) and collaborative system groups (CSGs) are often large systems with complex behavior. The complexity stems mainly from the interaction of the different components or subsystems (consider, for example, the case of several robots collaborating in pushing a door open or passing through a narrow passage). As a result, the behavior of CESs is difficult to completely model a priori. At the same time, CESs have to be continuously adapted and optimized to new runtime contexts (e.g., in the example of the collaborating robots, consider the case of an extra obstacle that makes the door harder to open).

Approach for online learning and adaptation of CESs and CSGs abstracted as black-box models

In this chapter, we present an approach for online learning and adaptation that can be applied in CESs and CSGs (but also other systems) that have (i) complex behavior that is unrealistic to completely model a priori, (ii) noisy outputs, and (iii) a high cost of bad adaptation decisions. We assume that the CES to be adapted is abstracted as a black-box model of the essential input and output parameters. Input parameters (knobs) can be set at runtime to change the behavior of the CES. Output parameters are monitored at runtime to assess whether the CES satisfies its goals. Noisy outputs refer to outputs whose values exhibit high variance, and thus may need to be monitored over long time periods. The cost of an adaptation decision (e.g., setting a new value for one of the knobs) refers to the negative impact of the adaptation decision on the CES.

Finding values of input parameters that optimize the outputs

Given the above assumptions, we focus on finding the values of the input parameters of a CES that optimize (maximize or minimize) its outputs. Our approach performs this optimization online—that is, while the system is running—and in several phases [Gerostathopoulos et al. 2018]. In doing so, it explores and exemplifies (i) how to build system models from observations of noisy system outputs; (ii) how to (re)use these models to optimize the system at runtime, even in the face of newly encountered situations; and (iii) how to incorporate the notion of cost of adaptation decisions in the above processes. Compared to related approaches, our approach focuses on providing statistical guarantees (in the form of confidence intervals and p-values) in different phases of the optimization process.

www.dbooks.org

15.2 A Self-Optimization Approach for CESs

A self-optimization approach for CESs must be (i) efficient in finding an optimal or close-to-optimal configuration fast, and (ii) safe in not incurring high costs of adaptation decisions. To achieve these goals, in our approach, we use prior knowledge of the system (the K in the MAPE-K loop for self-adaptive systems [Kephart and Chess 2003]) to guide the exploration of promising configurations. We also measure the cost of adaptation decisions in the optimization and stop the evaluation of bad configurations prematurely to avoid incurring high costs.

Self-optimization by finding the best configuration

Formally, the self-optimization problem we are considering consists of finding the minimum of a response or output function $f: X \rightarrow R$, which takes n input parameters $X_1, X_2, \ldots, X_n$, which range in domains $Dom(X_1), Dom(X_2), \ldots, Dom(X_n)$ respectively. X is the configuration space and corresponds to the Cartesian product of all the parameters' domains $Dom(X_1) \times Dom(X_2) \times \ldots \times Dom(X_n)$. A configuration C assigns a value to each of the input parameters.

Based on the definitions above, our approach for self-optimization of CESs relies on performing a series of online experiments. An experiment changes the value of one or more input parameters and collects values of the outputs. This allows us to assess the impact of the change to the input parameter on the outputs. The experiment-driven approach consists of the following three phases, also depicted in Figure 15-1 (where the CES is depicted in the upper right corner):

- Phase #1: Generation of system model
- Phase #2: Runtime optimization with cost handling
- Phase #3: Comparison with baseline configuration

These phases run consecutively; in each phase, one or more experiments are performed. An optimization round consisting of the three phases may be initiated via a human (e.g., an operator) or via the system itself, if the system is able to identify runtime situations where its behavior can be optimized. At the end of the optimization round, the system has learned an optimal or close-to-optimal configuration and decides (as part of phase #3) whether or not to use this instead of its current configuration.

The three phases are described below.

Using factorial analysis of variance to build knowledge models

The **"Generation of system model"** phase deals with building and maintaining the knowledge needed for self-optimization. Here, we use factorial analysis of variance (ANOVA) to process incoming raw data and automatically create a statistically relevant model that is used in the subsequent phases. This model describes the effect that

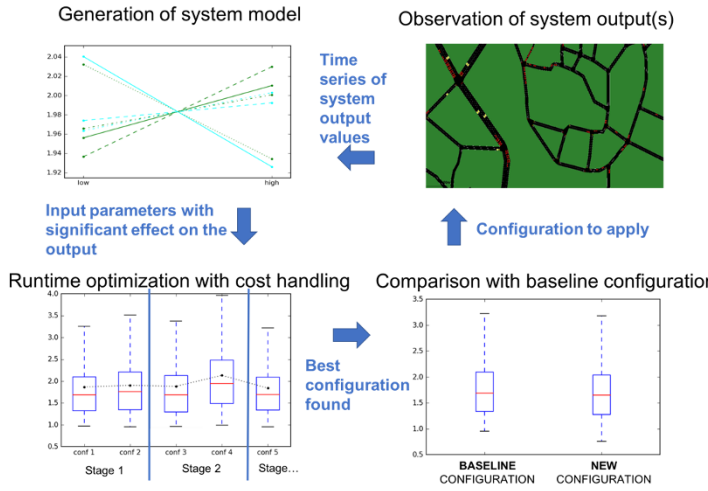


Fig. 15-1: *Overview of online experiment-driven learning and adaptation approach*

changing a single input parameter has on the output, while ignoring the effect of any other parameters. It also describes the effects that changing multiple input parameters together have on the output. This phase is run both prior to deploying the system using a simulator (to bootstrap the knowledge) and while the system is deployed in production using runtime monitoring (to gradually collect more accurate knowledge of the system in the real settings).Concretely, in the first step, the designer must discretize the domain of each input parameter in two or more values — this is an offline task. When the phase starts, in the second step, the system derives all the possible configurations given the parameter discretization (e.g., for three input parameters with two values each, it will derive 8 possible configurations capturing all possible combinations). This corresponds to a full factorial design in experimental design terminology [Ghosh and Rao 1996]. In the third step, for each configuration, an online experiment is performed and output values are collected. Once all experiments have been performed, between-samples factorial ANOVA is used to analyze the output datasets corresponding to the different configurations. The output of this phase is a list of input parameters ordered by decreasing effects (and corresponding significance levels) on the output.

The **"Runtime optimization with cost handling"** phase evaluates configurations via online experiments in a sequential way to find a configuration in which the system performs the best — that is, the output function is maximized or minimized. Instead of pre-designing

www.dbooks.org

the experiments to run as in phase #1, we use an optimizer that selects the next configuration to run based on the result of the previous experiment. In particular, the optimizer we have used so far employs Bayesian optimization with Gaussian processes [Shahriari et al. 2016]. The optimizer takes the output of phase #1—that is, a list of input parameters—as its input. For each parameter in the list, the optimizer selects a value from the parameter's domain (its original domain, not its discretized one used in phase #1) and performs an online experiment to assess the impact of the corresponding configuration on the system output. Based on the result of the online experiment, the optimizer selects another input parameter value, performs another online experiment, and so on. Before the start of the optimization process, the design sets the number of online experiments (iterations of the optimizer) that will be run in phase #2. The outcome of this phase is the best configuration found by the optimizer.

Using Bayesian optimization to find optimal configuration

We assume that configurations are rolled out incrementally in the system. If there is evidence that a configuration incurs high costs, its application stops and the optimizer moves on to evaluate the next configuration. So far, we assume that cost is measured in terms of the ratio of bad events — for example, complaints. Under this assumption, we use binomial testing to determine (with statistical significance) whether a configuration is not worth exploring anymore because of the cost overstepping a given threshold. A binomial test is a statistical procedure that tests whether, in a single sample representing an underlying population of two categories, the proportion of observations in one of the two categories is equal to a specific value. In our case, a binomial test evaluates the hypothesis that the predicted proportion of "bad events" issued is above a specific value — our "bad events" maximum threshold.

Using statistical testing to compare the optimal with the default configuration

"Comparison with baseline configuration" makes sure that a new configuration determined in the second phase is rolled out only when it is statistically significantly better than the existing configuration (baseline configuration). In order for the new configuration to replace the baseline configuration, checks must ensure that (i) it does indeed bring a benefit to the system (at a certain statistical significance level); and (ii) the benefit is enough to justify any disruption that may result from applying the new configuration to the system. The last point recognizes the presence of primacy effects, which pertain to inefficiencies caused to the users by a new configuration.

Concretely, in this phase, the effect of the (optimal) configuration output by phase #2 is compared to the default configuration of the system. This default configuration is provided offline by the system designers. To perform the comparison, the two configurations are rolled out in the system and values of the system output are collected. In other words, two online experiments are performed corresponding to the two configurations. Technically, the effect of the experiments is compared by means of statistical testing (so far, we have used t-tests) on the corresponding datasets of system outputs. This allows us to deduce whether the two configurations have a statistically significant difference (at a particular significance level *alpha*) in their effect on the system output.

15.3 Illustration on CrowdNav

Application of the approach to a traffic testbed

We illustrate our approach on the CrowdNav self-adaptation testbed [Schmid et al. 2017], whose goal is to optimize the duration of car trips in a city by adapting the parameters of the routing algorithm used for the cars' navigation. CrowdNav is released as an open-source project[1].

In CrowdNav, a number of cars are deployed in the German city of Eichstädt, which has approx. 450 streets and 1200 intersections. Each car navigates from an initial (randomly allocated) position to a randomly chosen destination in the city. When a car reaches its destination, it picks another one at random and navigates to it. This process is repeated forever.

To navigate from point A to point B, a car has to ask a router for a route (series of streets). There are two routers in CrowdNav: (i) the built-in router provided by SUMO (the simulation backend of CrowdNav) and (ii) a custom-built parametric router developed in our previous work. A certain number of cars ("regular cars") use the built-in router; the rest use the parametric router — we call these "smart cars."

The parametric router can be configured at runtime; it provides the seven configuration parameters depicted in Figure 15-2. Each parameter is an interval-scaled variable that takes real values within a range of admissible values, as provided by the designers of the system. Intuitively, certain configurations of the router's parameters yield better overall system performance.

[1] https://github.com/Starofall/CrowdNav

www.dbooks.org

Id	Name	Range	Description
1	route randomization	[0-0.3]	Controls the random noise introduced to avoid giving the same routes
2	exploration percentage	[0-0.3]	Controls the ratio of smart cars used as explorers[2]
3	static info weight	[1-2.5]	Controls the importance of static information (i.e., max. speed, street length) on routing
4	dynamic info weight	[1-2.5]	Controls the importance of dynamic information (i.e., observed traffic) on routing
5	exploration weight	[5-20]	Controls the degree of exploration of the explorers
6	data freshness threshold	[100-700]	Threshold for considering traffic-related data as stale and disregarding them
7	re-routing frequency	[10-70]	Controls how often the router should be invoked to re-route a smart car

Fig. 15-2: *Configurable (input) parameters in CrowdNav's parametric router*

Trip overhead is a prime example of noisy output

To measure the overall system performance, CrowdNav relies on the trip overhead metric. A trip overhead is a ratio-scaled variable whose values are calculated by dividing the observed duration of a trip by the theoretical duration of the trip — that is, the hypothetical duration of the trip if there were no other cars, the smart car travelled at maximum speed, and the car did not stop at intersections or traffic lights. Only smart cars report their trip overheads at the end of their trips (we assume that the rest of the cars act as noise in the simulation, so their effect can be observed only indirectly). Since some trips will have a larger overhead than others no matter what the router configuration is, the dataset of trip overheads exhibits high variance — it can thus be considered a noisy output.

Driver complaints model "bad events"

Together with the trip overhead, at the end of each trip, each smart car reports a complaint value — that is, a Boolean value indicating whether the driver is annoyed. The complaint value is generated based on the trip overhead and a random chance, so that some of the "bad trips" would generate complaints (but not all). To measure the cost of a bad configuration in CrowdNav, the metric of the complaint rate is used: the ratio of issued complaints to the total number of observed (trip overhead, complaint) tuples.

Finally, CrowdNav resides in different situations depending on two context parameters that can be observed, but not controlled: the number of regular (non-smart) cars and the number of smart cars. In particular, each context parameter can be in a number of predefined ranges. For example, the number of smart cars can be in one of the following ranges or states: 0-100, 100-200, 200-300, ..., 700-800, >800. All the possible situations are defined as the Cartesian product of the states of all context variables. In each situation, a different configuration might be optimal. The task of self-optimization in CrowdNav then becomes one of quickly finding the optimal configuration for the situation the system resides in and applying it.

In this context, quickly finding a configuration of parameters that minimizes the trip overhead in a situation, while keeping the number of complaints in check, entails understanding the effect a configuration has on both the trip overhead (the output we want to optimize for) and the complaint rate (the "bad events" metric).

Generalizing from this scenario, the problem to solve is as follows: "Given a set of input system parameters X, an output system parameter O with values exhibiting high variance, an environment situation S, and a cost parameter C, find the values of each parameter in X that optimize O in S without exceeding C, in the least number of attempts."

Our approach focuses the optimization on the important input parameters

We have evaluated the applicability of our experiment-driven self-optimization method on CrowdNav. Compared to performing optimization with all the input parameters (essentially skipping phase #1), our approach can reduce the optimization space, and consequently converge faster, by optimizing only the input parameters that have a strong effect on the output (trip overhead in the case of CrowdNav) [Gerostathopoulos et al. 2018].

15.4 Conclusion

In this chapter, we presented an approach for runtime optimization of CESs. Our approach relies on the concept of online experiments that consist of applying an adaptation action (changing a configuration) of a system that is running and observing the effect of the change on the system output. The approach consists of three stages that, together, combine optimization with statistical guarantees that come in the form of confidence intervals and observed effect sizes. We have applied the approach on a self-adaptation testbed where the routing of cars in a city is optimized at runtime based on tuning the

www.dbooks.org

configuration of the cars' parametric router. Our approach can be used in any system that can be abstracted as a black-box model of the essential input and output parameters.

15.5 Literature

[Gerostathopoulos et al. 2018] I. Gerostathopoulos, C. Prehofer, T. Bures: Adapting a System with Noisy Outputs with Statistical Guarantees. In: Proceedings of the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2018). ACM, 2018, pp. 58–68.

[Ghosh and Rao 1996] S. Ghosh, C. R. Rao, Eds.: Handbook of Statistics 13: Design and Analysis of Experiments, 1st edition. Amsterdam: North-Holland, 1996.

[Kephart and Chess 2003] J. Kephart, D. Chess: The Vision of Autonomic Computing In: Computer, vol. 36, no. 1, 2003, pp. 41–50.

[Schmid et al. 2017] S. Schmid, I. Gerostathopoulos, C. Prehofer, T. Bures: Self-Adaptation Based on Big Data Analytics: A Model Problem and Tool. In: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017). IEEE, 2017, pp. 102–108.

[Shahriari et al. 2016] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. de Freitas, Taking the Human Out of the Loop: A Review of Bayesian Optimization. In: Proceedings of the IEEE, vol. 104, no. 1, Jan. 2016, pp. 148–175.

[Sheskin 2007] J. Sheskin: Handbook of Parametric and Nonparametric Statistical Procedures, 4th ed. Chapman & Hall/CRC, 2007.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Diego Marmsoler, Technical University of Munich

16

Compositional Verification using Model Checking and Theorem Proving

Collaborative embedded systems form groups in which individual systems collaborate to achieve an overall goal. To this end, new systems may join a group and participating systems can leave the group. Classical techniques for the formal modeling and analysis of distributed systems, however, are mainly based on a static notion of systems and thus are often not well suited for the modeling and analysis of collaborative embedded systems. In this chapter, we propose an alternative approach that allows for the verification of dynamically evolving systems and we demonstrate it in terms of a running example: a simple version of an adaptable and flexible factory.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_16

www.dbooks.org

16.1 Introduction

Today more than ever, our daily life is determined by smart systems that are embedded into our environment. Modern systems even start collaborating with one another, making them collaborative embedded systems (CESs), which form collaborative system groups (CSGs). Due to the impact of such systems on modern society, verifying them has become an important task. However, their nature also imposes new challenges for verification.

Consider, for example, an adaptable and flexible factory as described in [Schlingloff 2018] and depicted in Figure 16–1. Here, robots transport items between machines and together they form a CSG with the common goal of producing a complex item from simpler items. During the lifetime of the CSG, new individual CESs (robots or maybe even machines) may join the group while others may leave it.

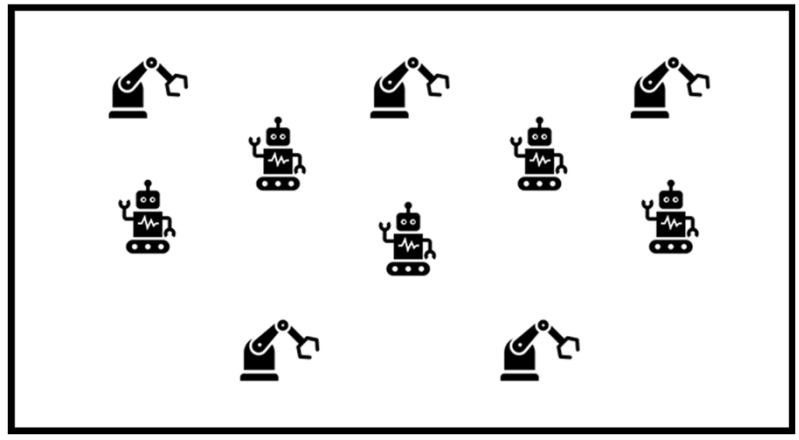


Fig. 16–1: *Smart production chain*

Since traditional verification techniques usually focus on static system structures, they reach their limit when it comes to the verification of CSGs. Thus, in the following, we describe a novel approach to the verification of such systems that allows us to consider dynamically evolving groups of systems using a combination of automatic and semi-automatic verification techniques.

In this chapter, we first describe the approach in more detail. We then demonstrate it by applying it to the verification of a simple adaptable and flexible factory. We conclude with a brief summary, discussion of limitations, and outlook.

16.2 Approach

Figure 16-2 depicts an overview of our approach for the compositional verification of a CSG (represented as a group of individual CESs in the center). Verification of a set of overall system properties (represented by the list at the top right of the figure) proceeds in three steps: (i) We first identify suitable contracts for the individual CESs (represented by the filled boxes). (ii) We then verify the individual CESs against their contracts (left part of the figure). (iii) Finally, we combine the individual contracts with the description of the architecture to verify overall system properties (right-hand part of the figure).

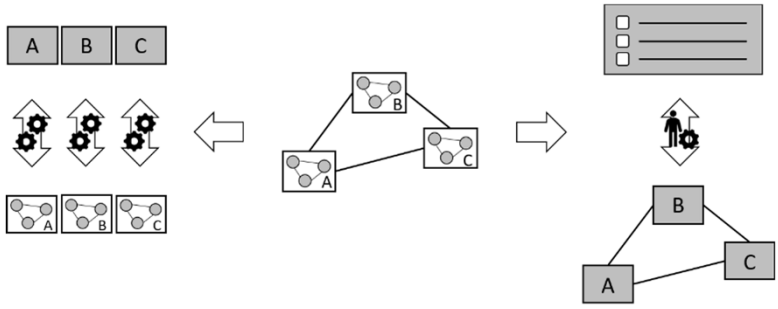


***Fig. 16-2**: Hybrid verification approach*

The approach is based on a formal system model that is based on FOCUS [Broy and Stolen 2012] and described in detail in [Marmsoler and Gleirscher 2016a], [Marmsoler and Gleirscher 2016b], and [Marmsoler 2019b]. To verify individual CESs against their contracts, we apply model checking [Clarke et al. 1986]. This allows us to change implementations of an individual CES and obtain fast feedback on whether the new implementation still satisfies the contracts of this CES.

Since we often do not know the exact number of CESs that participate in a CSG, we need to consider a possibly unlimited number of CESs. Thus, we apply interactive theorem proving [Nipkow et al. 2002] for the second step. The stability of results at the composition level justifies the additional effort that comes with interactive verification techniques compared to fully automatic techniques: as long as the single CESs satisfy their contracts, results at composition level remain valid.

To support a user in the development of specifications, the approach is implemented in terms of an Eclipse EMF-based modeling tool called FACTum Studio [Marmsoler and Gidey 2018], [Gidey et al.

www.dbooks.org

2019]. The tool allows a user to develop specifications and proof sketches using a combination of graphical and textual modeling techniques. The specification can then be used to generate corresponding models and verification conditions for both the nuXmv [Cavada et al. 2014] model checker and the interactive theorem prover Isabelle/HOL [Nipkow et al. 2002].

To further support the development of interactive proofs, the approach comes with a framework to support the verification of dynamic architectures implemented in Isabelle [Marmsoler 2018b], [Marmsoler 2019c].

16.3 Example

To demonstrate the approach, we apply it to verify a simple property for our smart production use case.

16.3.1 Specification

We first need to specify the data types for the messages exchanged between the systems of our CSG. Figure 16-3 depicts a corresponding specification in terms of an abstract data type [Broy et al. 1984]: it specifies a data type item to represent the items produced in the system. For our example, we assume that items depend on one another in the sense that the production of a certain item may require another item. To this end, we specify a relationship $\leq_{it}$ between items such that $item1 \leq_{it} item2$ means that the production of an $item2$ requires an $item1$. Note that the specification makes *items* an enumerable type, which allows us to use a successor function *succ* to obtain the successor of an item.

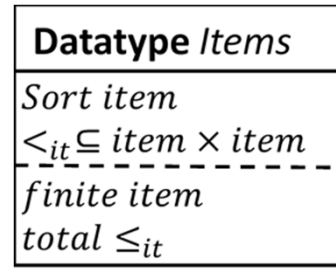


Fig. 16-3: *Production items*

As a next step, we have to specify the types of systems involved in our production chain. Figure 16-4 depicts a possible specification in terms of an architecture diagram [Marmsoler and Gidey 2019]: we specify two types of CES — machines and robots. Machines are parametrized by two items: one that represents the item a machine can produce, and one that represents the item the machine needs for the production. Thus, a system $Machine\langle item1, item2\rangle$ represents a machine that requires an $item1$ to produce an $item2$. A robot, on the

other hand, is parametrized by a single item that represents the item it is able to carry. For example, a system $Robot\langle item1\rangle$ is able to carry only items of type $item1$. In addition, the diagram requires that for every combination of items $it1$, $it2$, $it3$, where the production of $it3$ requires an $it2$ which in turn requires an $it1$, there is a machine $m1$ that can produce an item $it2$ when receiving an item $it1$ and a machine

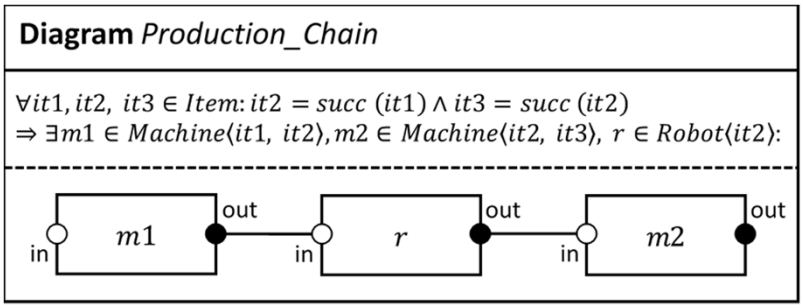


***Fig. 16-4**: Architecture diagram for a smart production chain*

$m2$ that requires an item $it2$ to produce an item $it3$, and a robot that can carry an item $it2$. Moreover, the diagram requires that the robot be connected to the machines via the correct ports, as depicted by the connections in the diagram.

Note that since we are using parameters here, the diagram actually specifies a production sequence of arbitrary length depending on the concrete items provided. Moreover, the specification allows individual CESs to leave and join the production chain as long as the architectural property is satisfied. For example, a robot may leave the CSG if there is another robot that can take over its responsibilities.

After specifying the architecture, we can specify the behavior of individual types of systems. Figure 16-5 depicts a simplified specification of a possible machine implementation in terms of a state machine: a machine waits for a source item before starting the production and delivering the item. In addition to the implementation, we must also specify contracts for the system using linear temporal logic [Manna and Pnueli 1992]. The contract specified for a machine in Figure 16-5, for example, states that whenever a machine obtains

www.dbooks.org

the required input item, it will eventually produce the desired output item.

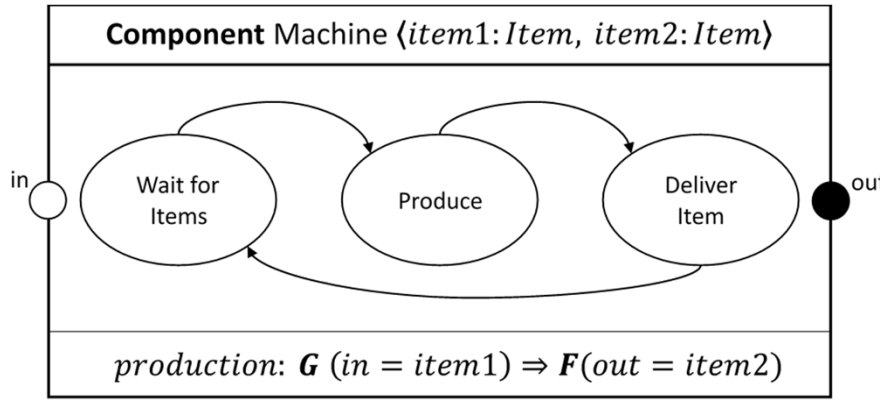


Fig. 16-5: *Specification of a machine*

Note that we use the machine's parameters *item*1 and *item*2 in formulating the contract.

Similarly, we have to specify the implementation of a robot, which is depicted in Figure 16-6: a robot collects an item, moves around, and finally drops the item when it reaches the correct position. Again, we have omitted details about the guards for the transitions for the sake of readability. And again, we also formulated a possible contract for a robot at the bottom of the diagram stating that a robot will always deliver a collected item.

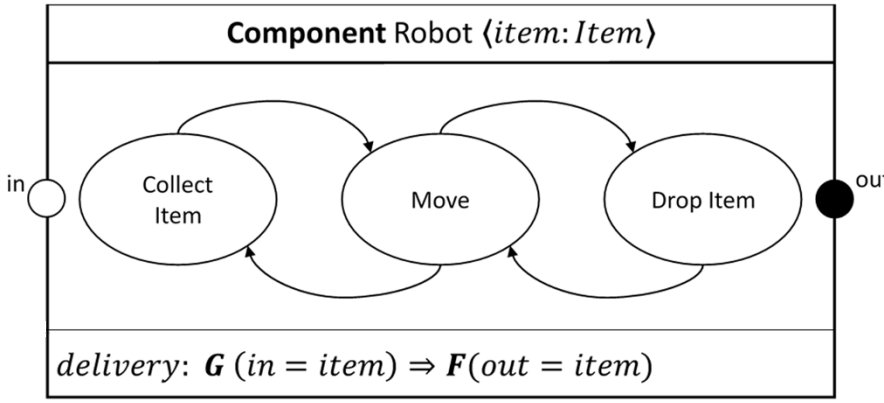


Fig. 16-6: *Specification of a robot*

16.3.2 Verification

Let us assume, for the purpose of our example, that we want to verify that the CSG can produce the final production item of a chain of arbitrary length, given that it is provided with the first item required in the chain. For example, if we are given a chain of items $item1 \leq_{it} item2 \leq_{it} \ldots \leq_{it} itemN$, then our group should be able to collaboratively produce item $itemN$ when it receives a corresponding $item1$.

As shown in Figure 16-2, verifying a specification of a CSG consists of two parts: first, we apply model checking to verify that a single component indeed satisfies its contracts. If we use FACTum Studio to model our system, we could then simply generate a model and corresponding verification conditions for the nuXmv model checker from the specification to automatically perform the verification.

Next, we have to combine the individual contracts to show that the overall system works correctly. To do so, we first show a smaller result that states that for every machine-robot-machine combination, when the first machine receives the correct input item, the second machine provides the correct output. Note that this involves combining three different contracts: the two contracts that ensure that the two machines function correctly, and another contract that ensures that the robot functions correctly. We can sketch this proof using an architecture proof modeling language (APML) [Marmsoler and Blakqori 2019], a notation similar to a sequence chart for sketching composition proofs. A possible APML proof sketch is shown in Figure 16-7: it first states the property in linear temporal logic at the top and then provides a proof sketch in the form of a sequence diagram. The proof sketch describes how the different contracts need to be combined to discharge the overall proof obligation.
Note the reference to the corresponding contracts: production, delivery, production.

Again, if we use FACTum Studio for the specification of the APML proof sketch, then we can automatically generate a corresponding proof for the interactive theorem prover Isabelle to check the soundness of the proof sketch.

www.dbooks.org

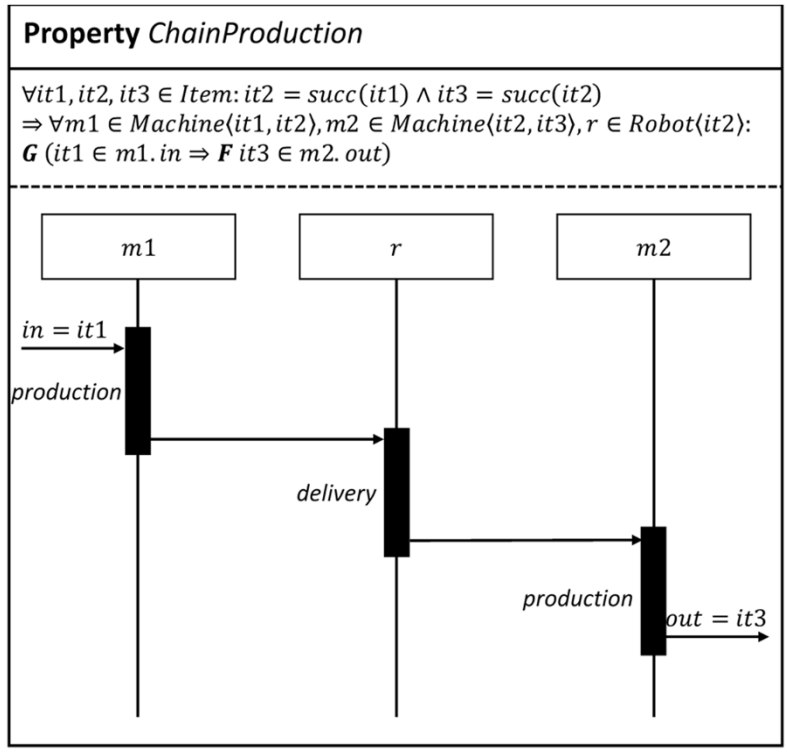


Fig. 16-7: *APML proof for smart production*

The result we just proved shows the correctness of one segment of our production chain. Now, to show the correctness of the complete chain, we have to repeat our argument for every segment of the chain. We can do this using a technique called *well-founded induction* [Winskel 1993]. The corresponding sketch is shown in Figure 16-8. This concludes the proof and therefore the verification of our production chain.

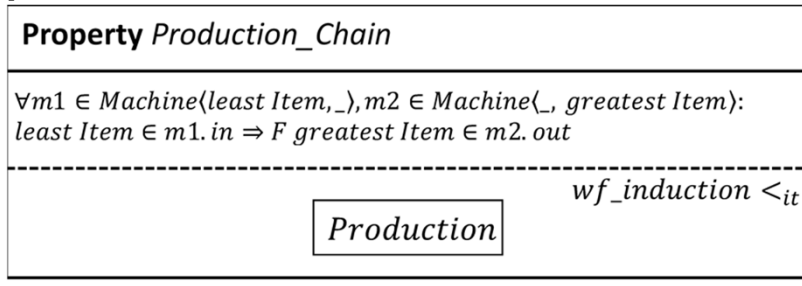


Fig. 16-8: *Well-founded induction over production chain*

16.4 Conclusion

In this chapter, we described an approach for verifying CSGs based on a combination of automatic and semi-automatic verification techniques and we demonstrated our approach in terms of a simple example. As shown by the example, the approach allows verification of CSGs that consist of an arbitrary number of individual CESs. Thus, it complements traditional verification approaches that usually assume a static structure with a fixed number of systems involved.

In addition to the example described in this chapter, the approach has been successfully applied to other domains as well, such as train control systems [Marmsoler and Blakqori 2019], architectural design patterns [Marmsoler 2018a], and even blockchain [Marmsoler 2019a]. While this showed the general feasibility of the approach, it also revealed some limitations: one weakness concerns the expressive power of our contracts. As of now, contracts are limited to a restricted form of linear temporal logic and many interesting properties cannot be expressed. Thus, future works should investigate alternative notions of contracts to increase expressiveness. Another weakness concerns the generation of Isabelle proofs from APML proof sketches. Sometimes, the proofs generated do not contain all the necessary details required by Isabelle to confirm the proof and some manual additions may be necessary. Thus, future work should also investigate possibilities to generate more complete proofs to minimize interactions with the interactive theorem prover. Finally, our system model assumes the existence of a global time to synchronize different components. While this assumption is suitable for some scenarios, there might be other scenarios where it might not hold. Thus, future work should investigate possibilities to weaken this assumption.

16.5 Literature

[Broy et al. 1984] M. Broy, M. Wirsing, C. Pair: A Systematic Study of Models of Abstract Data Types. In: Theoretical Computer Science, vol. 33, 1984, pp. 139-174.

[Broy and Stolen 2012] M. Broy, K. Stolen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement, Springer Science & Business Media, 2012.

[Cavada et al. 2014] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta: The nuXmv Symbolic Model Checker. In: Biere A., Bloem R. (eds) Computer Aided Verification. CAV 2014.

[Clarke et al. 1986] E. M. Clarke, E. A. Emerson, A. P. Sistla: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. In: ACM Trans. Program. Lang. Syst., vol. 8, 4 1986, pp. 244–263.

www.dbooks.org

[Gidey et al. 2019] H. K. Gidey, A. Collins, D. Marmsoler: Modeling and Verifying Dynamic Architectures with FACTum Studio. In: Arbab F., Jongmans SS. (eds) Formal Aspects of Component Software. FACS 2019.

[Manna and Pnueli 1992] Z. Manna, A. Pnueli: The Temporal Logic of Reactive and Concurrent Systems, Springer New York, 1992.

[Marmsoler and Gleirscher 2016a] D. Marmsoler, M. Gleirscher: Specifying Properties of Dynamic Architectures Using Configuration Traces. In: International Colloquium on Theoretical Aspects of Computing, Springer, 2016, pp. 235–254.

[Marmsoler and Gleirscher 2016b] D. Marmsoler, M. Gleirscher: On Activation, Connection, and Behavior in Dynamic Architectures. In: Scientific Annals of Computer Science, vol. 26, 2016, pp. 187–248.

[Marmsoler 2018a] D. Marmsoler: Hierarchical Specification and Verification of Architecture Design Patterns. In: Fundamental Approaches to Software Engineering - 21th International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, 2018.

[Marmsoler and Gidey 2018] D. Marmsoler, H. K. Gidey: FACTUM Studio: A Tool for the Axiomatic Specification and Verification of Architectural Design Patterns. In: Formal Aspects of Component Software - FACS 2018 - 15th International Conference, Proceedings, 2018.

[Marmsoler 2018b] D. Marmsoler: A Framework for Interactive Verification of Architectural Design Patterns in Isabelle/HOL. In: The 20th International Conference on Formal Engineering Methods, ICFEM 2018, Proceedings, 2018.

[Marmsoler and Blakqori 2019] D. Marmsoler, G. Blakqori: APML: An Architecture Proof Modeling Language. In: Formal Methods – The Next 30 Years, Cham, 2019.

[Marmsoler 2019a] D. Marmsoler: Towards Verified Blockchain Architectures: A Case Study on Interactive Architecture Verification. In: Formal Techniques for Distributed Objects, Components, and Systems, Cham, 2019.

[Marmsoler 2019b] D. Marmsoler: A Denotational Semantics for Dynamic Architectures. In: 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE), 2019.

[Marmsoler 2019c] D. Marmsoler: A Calculus for Dynamic Architectures. In: Science of Computer Programming, vol. 182, 2019, pp. 1-41.

[Marmsoler and Gidey 2019] D. Marmsoler, H. K. Gidey: Interactive Verification of Architectural Design Patterns in FACTum. In: Formal Aspects of Computing, vol 31, 2019, pp. 541-610.

[Nipkow et al. 2002] T. Nipkow, L. C. Paulson, M. Wenzel: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, vol. 2283, Springer Science & Business Media, 2020.

[Schlingloff 2018] B. Schlingloff: Specification and Verification of Collaborative Transport Robots, in 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), 2018.

[Winskel 1993] Glynn Winskel: The Formal Semantics of Programming Languages: An Introduction. MIT Press, Cambridge, MA, USA, 1993.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Steffen Hillemacher, RWTH Aachen University
Nicolas Jäckel, FEV Europe GmbH
Christopher Kugler, FEV Europe GmbH
Philipp Orth, FEV Europe GmbH
David Schmalzing, RWTH Aachen University
Louis Wachtmeister, RWTH Aachen University

17

Artifact-Based Analysis for the Development of Collaborative Embedded Systems

One of the major challenges of heterogeneous tool environments is the management of different artifacts and their relationships. Artifacts can be interdependent in many ways, but dependencies are not always obvious. Furthermore, different artifact types are highly heterogeneous, which makes tracing and analyzing their dependencies complicated. As development projects are subject to constant change, references to other artifacts can become outdated. Artifact modeling tackles these challenges by making the artifacts and relationships explicit and providing a means of automated analysis. We present a methodology for artifact-based analysis that enables analysis of heterogeneous tool environments for architectural properties, inconsistencies, and optimizations.

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_17

www.dbooks.org

17.1 Introduction

Consistency of artifacts during the development of collaborative embedded systems

The development of collaborative embedded systems (CESs) typically involves the creation and management of numerous interdependent development artifacts. Requirements documents specify, for example, all requirements that a system under development must fulfil during its lifetime, whereas system architectures written in the Systems Modeling Language (SysML) [SysML 2017] enable system architects to describe the logical and technical architecture of the system. If the expected behavior of a system and its system components is also modeled in SysML, automatically generated test cases [Drave et. al. 2019] can be used to check the system for compliance with these system requirements. Accordingly, the creation of these development artifacts extends through all phases of system development and thus over the entire project duration. Consequently, different developers create system requirements, architecture, and test artifacts using diverse tools of the respective application domain. Therefore, all artifacts must be checked for consistency, especially if further development artifacts are to be generated automatically in a model-driven approach. For example, it must be ensured that all components that are mentioned in the system requirements or for which system requirements exist are also present in the system architecture, or that all values checked by a test case match the respective target values specified in a requirement.

Heterogeneous development tools

Another challenge that arises during the system development process for CESs is the use of different tools during different stages of the development project. As CESs aim to connect different embedded systems handling multiple tasks in different embedding environments, heterogeneous tools adapted to the application domain are also used to create them. Furthermore, practice has shown that new tools are introduced to the project and obsolete tools are replaced by new ones to meet the challenges that arise in different development phases whenever insuperable tool boundaries are reached. As a result, the project becomes more complex, as new tools create new dependencies and other relationships, a situation that is amplified by the fact that the number of artifacts and their interdependence during development constantly increases. Since these various development tools are often incompatible with each other and do not support relationship validation across tool

interfaces, we use artifact-based analyses to enable automatic analysis of relationships and architectural consistency.

Artifact-based analysis

To tackle this challenge, automating artifact-based analysis enables the system developers to model the artifacts created during a project and to automatically analyze their relationships and changes. Artifact-based modeling and analysis were originally developed for software projects [Greifenberg et. al. 2017], but with slight modifications, also offer a decisive advantage in systems projects [Butting et. al. 2018]. For this purpose, we introduce a project-specific artifact model that is adapted to the individual project situation and thus unambiguously models the artifacts that occur in the project and illustrates their relationships.

Application of artifact-based analysis

We show the application of artifact-based analysis using the example of DOORS Next Generation (Doors NG) and Enterprise Architect (EA). To this end, we create an artifact model that models the structure and the elements of the exports of Doors NG and EA, as well as their relationships. We then describe the extraction of the structures and prepare the extracted data for further processing by analysis. For this purpose, we have developed static extractors that convert the exports into artifact data (object diagrams). Finally, we model analyses using Object Constraint Language (OCL) expressions over the artifact metamodel and show the execution of corresponding analyses on the extracted data.

17.2 Foundations

In this section, we present the modeling languages and model-processing tools used in our approach and explain how to use these to describe artifacts and artifact relationships.

UML/P

A language profile of UML

The UML/P language family [Rumpe 2016], [Rumpe 2017] is a language profile of the Unified Modeling Language (UML) [UML 2015]. Due to the large number of languages involved, their fields of application, and the lack of formalization, UML is not directly suitable for model-driven development (MDD). However, it could be made suitable by restricting the modeling languages and language constructs allowed, as has been done in the UML/P language family. A textual version of UML/P that can be used in MDD projects was developed in [Schindler 2012]. The approach for the artifact-based

www.dbooks.org

analysis of MDD projects uses the languages Class Diagram (CD), Object Diagram (OD), and OCL.

Class Diagrams in UML/P

Class diagrams for analysis

Class diagrams serve to represent the structure of software systems and form the central element for modeling software systems with UML. CDs are primarily used to introduce classes and their relationships. In addition, they can be used to model enumerations and interfaces, associated properties such as attributes, modifiers, and method signatures, as well as various types of relationships and their cardinalities. CDs can be used in analysis to structure concepts of the problem domain, in addition to being utilized to represent the technical, structural view of a software system—that is, as the description of source code structures [Rumpe 2016]. For this use case in particular, [Roth 2017] developed an even more restrictive variant of the UML/P class diagrams: the language Class Diagram for Analysis (CD4A). In the approach presented here, CD4A is used to model structures in model-based development projects.

Object Diagrams in UML/P

Object diagrams for representing problem domain concepts

Object diagrams are suitable for specifying exemplary data of a software system. They describe a state of the system at a concrete point in time. ODs may conform to the structure of an associated class diagram. Checking whether an object diagram corresponds to the predefined structure of a class diagram is generally not trivial. For this reason, [Maoz et. al. 2011] describes an approach for an Alloy-based [Jackson 2011] verification technique. In object diagrams, objects and the links between objects are modeled. The object state is modeled by specifying attributes and assigned values. Depending on the intended use, object diagrams can describe a required situation of the software system or represent a prohibited or existing situation of the software system. The current version of the UML/P OD language allows the definition of hierarchically nested objects in addition to the concepts described in [Schindler 2012]. This has the advantage that hierarchical relationships can also be displayed as such in the object diagrams. In this work, CDs are not used to describe the classes of an implementation, but when used for descriptions on a conceptual level, objects of associated object diagrams also represent concepts of the problem domain instead of objects of a software system. In our approach, object diagrams are used to describe analysis data — that is, they reflect the current state of the project at the conceptual level.

OCL

OCL for analysis

OCL is a specification language of UML that allows additional conditions of other UML languages to be modeled. For example, OCL can be used to specify invariants of class diagrams, conditions in sequence diagrams, and to specify pre- or post-conditions of methods. The OCL variant of UML/P (OCL/P) is a Java-based variant of OCL. Our approach uses the OCL/P variant only. OCL is used only in conjunction with class diagrams throughout this approach. OCL expressions are modeled within class diagram artifacts.

17.3 Artifact-Based Analysis

This section provides an overview of the solution concept developed for performing artifact-based analyses and is largely based on the work published in [Greifenberg 2019]. Before we present the analyses in more detail, let us define the terms artifact, artifact model, and artifact data.

Definition 17-1*: Artifact*

An artifact is an individually storable and uniquely named unit serving a specific purpose in the context of a development process.

Artifact definition

This definition focuses more on the physical manifestation of the artifact rather than its role in the development process. It is therefore less restrictive than the level characterization presented in [Fernández et. al. 2019]. Furthermore, the definition requires an artifact to be stored as an individual, referenceable unit. Nonetheless, an artifact must serve a specific purpose within a development process, making its creation and maintenance otherwise obsolete. On the other hand, the definition does not enforce restrictions on the integration of the artifact into the development process — that is, an artifact does not necessarily have to be an input or output of a certain process step. Artifacts may also exist only as intermediate or temporary contributions of a tool chain. Moreover, the definition largely ignores the logical content of artifacts. This level of abstraction enables an effective analysis of the artifact structure taking the existing heterogeneous relationships into account instead of analyzing the internal structure of artifacts.

www.dbooks.org

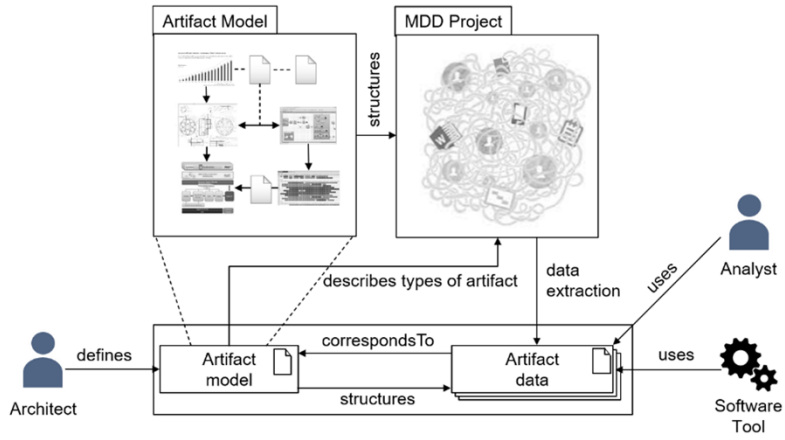


Fig. 17–2: *The role of an artifact model and corresponding artifact data within an MDD project*

Role of an artifact model and artifact data within an MDD project

An important part of the overall approach is the identification of the artifacts, tools, systems, etc. present in the development process and their relationships. Different modeling techniques provide a means to make these explicit and thus enable model-based analyses. Figure 17-2 gives an overview of the model-based solution concept. First, the types of artifacts, tools, and other elements of interest, as well as their relationships within a development process, must be defined. It is the task of an architect, who is well-informed about the entire process, to model these within an artifact model (AM). This model structures the artifact landscape of the corresponding process or a development project. The AM defines only the types of elements and relationships and not the specific instances; therefore, this model can remain unchanged over the entire life cycle of the process or the project unless new types of elements or relationships are added or removed. Moreover, once created, the model can be reused completely or partially for similar projects.

Definition 17-3*: Artifact model*

The artifact model defines the relevant artifact types and the associated relationship types of a development process to be examined.

Specific instances of an AM are called artifact data and reflect the current project status. Ideally, artifact data can be extracted automatically and saved in one or more artifacts.

Definition 17-4: *Artifact data*

Artifact data contains information about the relevant artifacts and their relationships that exist at a specific point in time in an engineering process. Artifact data are instances of a specific artifact model.

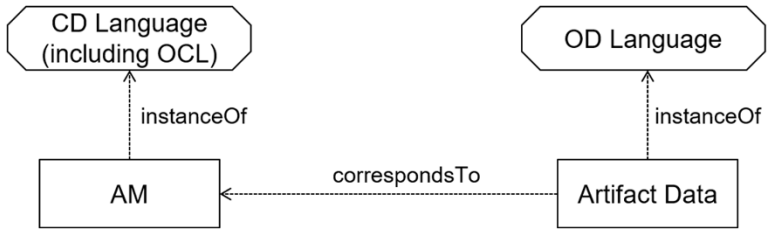


***Fig. 17–5**: Modeling languages used for the artifact model and data*

Relationship of artifact data to an artifact model

Artifact data are in an ontological instance relationship [Atkinson and Kuhne 2003] to the AM. Each element and each relationship from the artifact data correspond to an element or a relationship type of the AM. The AM thus prescribes the structure of its artifact data. Figure 17-5 shows how this is achieved in terms of modeling techniques. During the artifact-based analyses, artifact data represent the project state at a certain point in time. Analysts and analysis tools use the artifact data to understand the current project state, to check certain relationships, create reports, and to check for optimization potential within the project. Ultimately, the goal is to make the software development process as efficient as possible. This approach is especially suited for checking the consistency of the architecture of model-driven software development projects or processes. It is capable of handling input models, model-driven development (MDD) tools—which themselves consist of artifacts—and handwritten or generated artifacts that belong to the end product of the development process. In such a case, the AM depends on the languages, tools, and technologies used in the development process or project. Thus, it is usually tailored specifically to a process or project.



***Fig. 17–6**: Steps for enabling artifact-based analysis*

www.dbooks.org

Enabling artifact-based analysis

In order to perform artifact-based analyses as shown in Figure 17-6, the first step is to create a project-specific AM. Once created, analyses based on the artifact data are specified. Finally, after the two previous steps, the artifact-based analysis can be performed.

Artifact Model Creation

Creation of an artifact model

The first step of the methodology is the creation of an AM. The AM determines the scope for specific analyses based on the corresponding artifact data. It explicitly defines the relationships between the artifacts and specifies prerequisites for the analyses. Additionally, using the CD and OCL languages, model-driven development tools can be used to analyze the artifact data. [Greifenberg 2019] presents an AM core and a comprehensive AM for model-driven development projects. If a new AM has to be created, or an existing AM has to be adapted, the AM core and parts of existing project-specific AMs should be reused. A methodology for this can also be found in [Greifenberg 2019].

Types of artifacts as central elements of an artifact model

The central elements of any AM are the types of artifacts modeled. All project-specific types of files are eligible to be contained in the AM. Artifacts can contain each other. Typical examples of artifacts that contain other artifacts are archives or folders in the file system. However, database files or models containing artifacts are also possible. Figure 17-7 shows the relevant part of the reusable AM core as presented in [Greifenberg 2019].

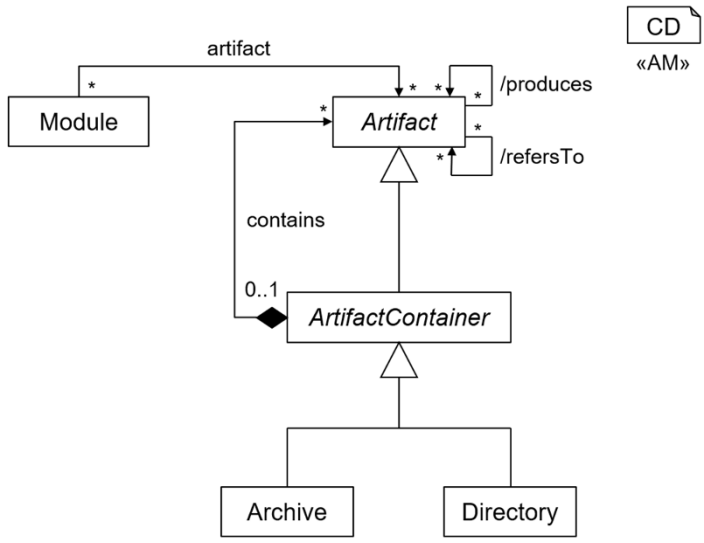


Fig. 17–7: *Reusable artifact model core as presented in [Greifenberg 2019]*

Composition of artifacts and artifact containers

In this part of the AM core, the composite pattern [Gamma et. al. 1995] ensures that archives and folders can contain each other in any order. Each type of artifact is contained in exactly one artifact container. If all available artifact types are modeled, there is exactly one type of artifact not contained by a container — that is, the root directory of the file system. Furthermore, artifacts can contribute to the creation of other artifacts (creates relationship) and they can statically refer to other artifacts (refers to relationship). These artifact relationships are defined as follows:

Definition 17-8*: Artifact reference*

If an artifact needs information from another artifact to fulfil its purpose, then it refers to the other artifact.

Definition 17-9*: Artifact contribution*

An existing artifact contributes to the creation of the new artifact (to its production) if its existence and/or its contents have an influence on the resulting artifact.

Refining an artifact model in project-specific extensions

Both relationships are defined in the AM as a derived association. Therefore, it is vital to specify these relationships further in project-specific AMs, while it is already possible to derive artifact data analyses from these associations. The specialization of associations is defined using OCL conditions [Greifenberg 2019], since the CD language is not suitable for this.

Specification of Artifact Data Analysis

Specifying analysis of artifact data

The second step of the methodology is the specification of project-specific analyses that are based on the AM created in the first step. These analyses must be repeatable and automated. They can be implemented either by one person, an analyst or analysis tool developer, or an analyst can specify the analyses as requirements for the analysis tool developer, who then implements an appropriate analysis tool. In this work, analyses are specified using OCL:

1. The CD language—used to model the AM—and OCL are well suited for use in combination to define analyses.

www.dbooks.org

2. OCL has already been used to define project-specific analyses in [Greifenberg 2019]. Reusing familiar languages and providing example analyses shortens the learning curve for analysts.
3. OCL has mathematically sound semantics that enable precise analyses. Moreover, OCL expressions are suitable as input for a generator that can automatically convert them into MDD tools, thus reducing the effort for the developer of the analysis tool.

Artifact-Based Analyses

Executing artifact-based analysis

The third step in Figure 17-6 is the artifact-based analysis, which executes the previously specified analyses. This step is refined into five sub-steps. Each step is supported by automated and reusable tools. Figure 17-10 presents these steps and the corresponding tools.

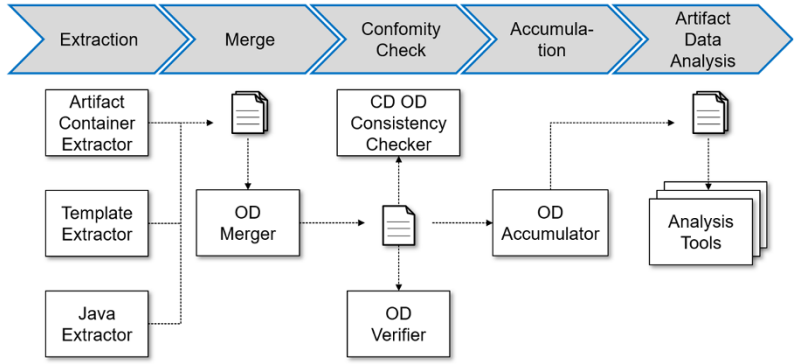


***Fig. 17–10**: Steps of artifact-based analysis with tools (rectangles), resulting files (file symbols), and the execution flow (directed arrows)*

Steps and components for performing artifact-based analysis

The first step in artifact-based analyses is to extract relevant project data. If stored in different files, the data must be merged. The entire data set is then checked for compliance with the AM. In the next step, the data is accumulated based on the specification of the AM, to ensure the derived properties are present for the last step, the artifact data analysis. [Greifenberg 2019] presents a tool chain that can be used to collect, merge, validate, accumulate, and finally, to analyze artifact data. The tool chain supports all sub-steps of the artifact-based analysis. The individual steps are each performed by one or more small tools that, combined, form the tool chain. The tools shown in Figure 17-10 are arranged according to the order of execution of the tool chain. The architecture as a tool chain is modular and adaptable. The primary data format for exchanging information between tools is

object diagrams. New tools can be added without having to adjust other tools. Existing tools can be adjusted or removed from the tool chain without the need to adjust other tools. Therefore, when using the tool chain in a new project, project-specific adjustments usually have to be made. The architecture chosen supports the reuse and adaptation of individual tools.

17.4 Artifact Model for Systems Engineering Projects with Doors NG and Enterprise Architect

Artifact-based analysis of Doors NG and Enterprise Architect

To demonstrate the practicability of the artifact-based approach, this section describes an example of artifact-based analysis of systems engineering projects with textual requirements and logical architecture components in SysML. Doors NG and Enterprise Architect are commonly used tools for these purposes. Doors NG enables engineers to define and maintain requirements in a collaborative development environment. Enterprise Architect is a solution for modeling, visualizing, analyzing, and maintaining systems and their architectures. Standards, such as UML and SysML, are supported. In our example, we focus on the definition of requirements in Doors NG and the modeling of systems and their components in Enterprise Architect. Here, system components are modeled with Internal Block Diagrams (IBD) and corresponding Block Definition Diagrams (BDD) from the SysML standard.

17.4.1 Artifact Modeling of Doors NG and Enterprise Architect

Creation of an artifact model for Doors NG and Enterprise Architect

The creation of the artifact model for this example includes the identification of artifact types used in the project as a first step. Since, in this example, we consider two tools whose files cannot be read directly via an open standard, suitable exchange formats must first be identified. The XML-based XMI exchange format, which is supported by Enterprise Architect as a tool-independent exchange format, is therefore taken as the exchange format for Enterprise Architect. Furthermore, a ReqIF export is used for the cloud-based data format of Doors NG for information exchange, which also enables a cross-tool exchange of requirements. The challenge here is that the requirements stored in the development tools are no longer present as individually stored units, but rather as what are referred to as artifact containers (cf. Figure 17-7), in which several development artifacts— which must first be identified and extracted for subsequent

www.dbooks.org

analysis—have been combined. Once these basic artifact types (named "EA Export" and "Doors Export" in the artifact model in Figure 17-11) have been identified, the relevant information contained in the exports must be modeled and related.

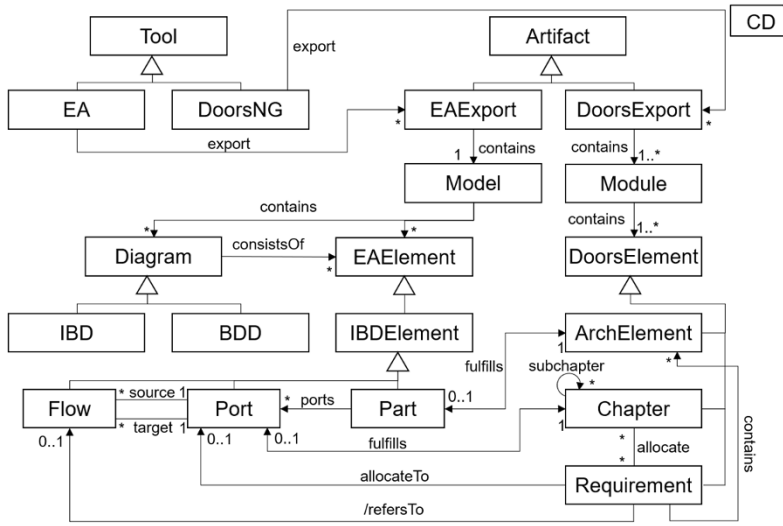


***Fig. 17-11**: Artifact model for exports of Doors NG and Enterprise Architect, as well as their relationships*

Modeling exports of Doors NG and Enterprise Architect

In the XMI export created by Enterprise Architect, exactly one model for the overall system modeled in the EA project is exported. This model contains any number of diagrams and elements (named *Diagram* and *EAElement* in the artifact model of Figure 17-11). Furthermore, each diagram has any number of elements, represented in the class diagram of the artifact model under consideration by the *consistsOf* association. Since the example considered is limited to architectural elements, not all diagram types of SysML are modeled in the artifact model; only the structural diagrams relevant for the logical architecture are modeled in the form of the Internal Block Diagram (IBD) and the Block Definition Diagram (BDD). A decisive advantage of the artifact models is that not all possible artifacts have to be modeled; the model can be limited to the artifacts relevant for the analysis. Similarly, only signal flows and parts—as the internal representation of ports in EA—are defined for the example under consideration. In the BDD, only the block is modeled as a relevant diagram element and all relationships in the BDD are no longer displayed. The ReqIF export of Doors NG is also represented as an artifact in the artifact model. Each *DoorsExport* contains at least one,

but otherwise any number of modules that also contain one or more *DoorsElements*. In this context, a mixture of *Chapters*, *Requirements,* and *ArchElements* serve as specialized *DoorsElements*.

17.4.2 Static Extractor for Doors NG and Enterprise Architect Exports

Static extraction of artifact data

To verify automatically that the current project complies with the architecture defined, all elements of the artifact model must be loaded from the two exports. To achieve this, we implemented static extractors, which parse the exports and load relevant information into our internal representation. For this purpose, the extractor transforms relevant data into an object diagram — that is, the artifact data. This workflow is shown in Figure 17-12. The artifact data extracted from the tool exports is tool-specific at first and needs further consolidation. This means that tool-specific artifact data is merged into a consistent data set (object diagram): the artifact data of the system. During this step, associations between objects of different diagrams are constructed (extracted from name references) and objects of the same type and name are merged automatically. Relationships between elements of different exports are constructed during this step. The resulting object diagram gives a view of the current project architecture and enables analysis.

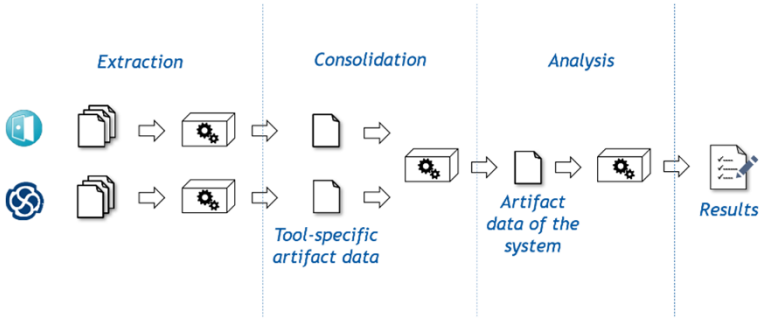


Fig. 17–12: *Tool chain workflow from artifact data extraction to analysis*

17.4.3 Analysis of the Extracted Artifact Data

Analysis of extracted artifact data

After the extraction and consolidation of artifact data, artifact-based analyses can be defined on the previously constructed project-specific artifact model and executed on the merged and consolidated artifact data. However, analyses are executed only on artifact data that conforms to the artifact model. Therefore, in a first step, the tool chain

www.dbooks.org

checks whether the artifact data is an instance of the artifact model and executes further well-formedness constraints. If the merged artifact data is well-formed and conforms to the artifact model, then defined analyses are executed on the artifact data. We model analyses as constraints of OCL over the defined artifact model. This enables us to define analyses without deeper programming experience and to execute analyses automatically on the extracted data without having knowledge of the internal data structure of the analysis tool. To this end, our tool chain transforms modeled analyses into machine code and executes this code on the internal representation of the artifact data.

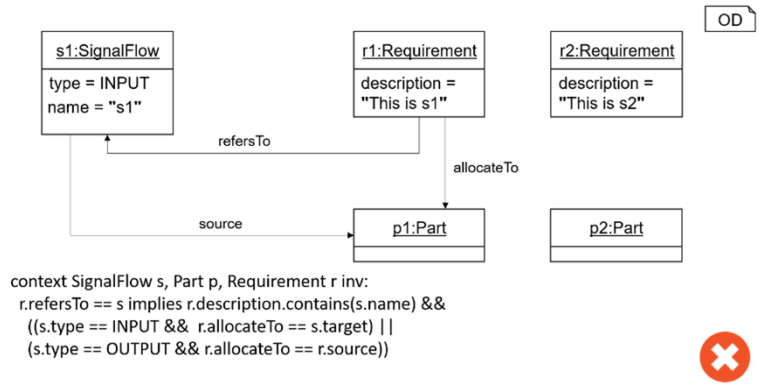


Fig. 17–13: *Example of artifact data invalidating a defined analysis constraint*

An example of modeling analysis using OCL constraints

An example of extracted artifact data invalidating an OCL analysis constraint is given in Figure 17-13. The constraints define that the name in the description of a requirement matches the name of a signal flow the requirement refers to, and that the part allocated to the requirement must be the target of this signal flow. In the artifact data extracted, however, the part *p1* allocated to the requirement *r1* is the source of the referred signal flow *s1*. The execution failure of the analysis is noted in the analysis report and implies required changes for project well-formedness. Changing the part *p1* to be the target of signal flow *s1* instead of its source validates the analysis as shown in Figure 17-14**.** The automated test in both sources checks that the models are consistent in both tools during the whole development process. The check throws an error if an inconsistency occurs, thus notifying developers of potential problems.

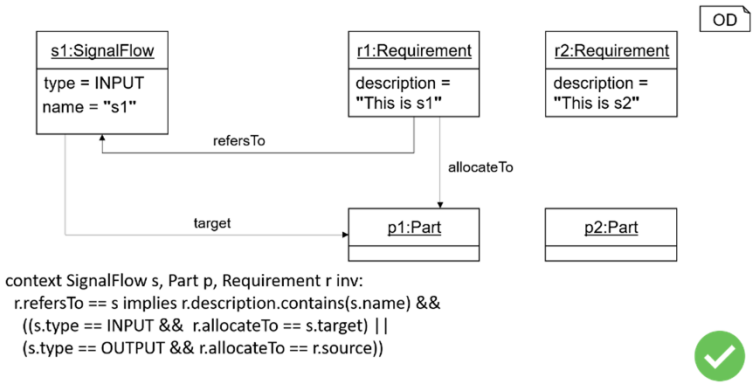


Fig. 17–14: *Example of artifact data validating a defining analysis constraint*

17.5 Conclusion

Employing artifact-based analysis to facilitate model-driven development

Model-driven development aims to reduce the complexity in the development of collaborative embedded systems by reducing the conceptual gap between problem and solution domain. The use of models and MDD tools enables at least a partial automation of the development process. In larger development projects involving several different domains in particular, the huge number of different artifacts and their relationships makes managing them difficult. This can lead to poor maintainability or an inefficient process within the project. The goal of the approach presented is the development of concepts, methods, and tools for artifact-based analysis of model-driven software development projects. Here, the artifact-based analysis describes a reverse engineering methodology that enables repeatable and automated analyses of artifact structures. In this approach, UML/P provides the basis for modeling artifacts and their relationships, as well as specifying analyses. A combination of the UML/P class diagrams and OCL is used to create project-specific artifact models. Additionally, analysis specifications can be defined using OCL while artifact data that represents the current project state is defined using object diagrams, which are instances of the artifact model. This allows the consistency between an AM and its artifact data to be checked. The models are specified in a human-readable form but can also be processed automatically by other MDD tools. The example presented for artifact-based analysis of Enterprise Architect and Doors NG shows the practicability for checking the consistency of artifacts across heterogeneous tools. Here, automated analyses enable system architects to check the conformity of specified components to

www.dbooks.org

requirements and enable requirement engineers to trace the impact of changes on the specified architecture.

17.6 Literature

[Atkinson and Kuhne 2003] C. Atkinson, T. Kuhne: Model-Driven Development: A Metamodeling Foundation. In: IEEE Software, 2003, pp. 36–41.

[Atzori et. al. 2010] L. Atzori, A. Iera, G. Morabito: The Internet of Things: A Survey. In: Computer Networks, 2010, pp. 2787 – 2805.

[Brambilla et. al. 2012] M. Brambilla, J. Cabot, M. Wimmer: Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers, 2012.

[Butting et. al. 2018] A. Butting, T. Greifenberg, B. Rumpe, A. Wortmann: On the Need for Artifact Models in Model-Driven Systems Engineering Projects. In: Software Technologies: Applications and Foundations, Springer, 2018, pp. 146-153.

[Cheng et. al. 2015] B. H. C. Cheng, B. Combemale, R. B. France, J. Jézéquel, B. Rumpe: On the Globalization of Domain-Specific Languages. In: Globalizing Domain-Specific Languages. LNCS 9400, Springer, 2015, pp 1–6.

[Drave et. al. 2019] I. Drave, T. Greifenberg, S. Hillemacher, S. Kriebel, E. Kusmenko, M. Markthaler, P. Orth, K. S. Salman, J. Richenhagen, B. Rumpe, C. Schulze, M. von Wenckstern, A. Wortmann: SMArDT Modeling for Automotive Software Testing. In: R. Buyya, J. Bishop, K. Cooper, R. Jonas, A. Poggi, S. Srirama: Software: Practice and Experience. 49(2), Wiley Online Library, 2019, pp. 301-328.

[Ebert and Favaro 2017] C. Ebert, J. Favaro: Automotive Software. In: IEEE Software, Vol. 34, 2017, pp. 33-39.

[Fernández et. al. 2019] D.M. Fernández, W. Böhm, A. Vogelsang, J. Mund, M. Broy, M. Kuhrmann, T. Weyer, 2019. Artefacts in Software Engineering: A Fundamental Positioning. In: Software & Systems Modeling, 18(5), pp. 2777-2786.

[France and Rumpe 2007] R. France, B. Rumpe: Model-Driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering (FOSE '07), 2007, pp. 37-54

[Gamma et. al. 1995] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[Greifenberg 2019] T. Greifenberg: Artefaktbasierte Analyse modellgetriebener Softwareentwicklungsprojekte. In: Aachener Informatik-Berichte, Software Engineering, Band 42, Shaker Verlag, 2019 (available in German only).

[Greifenberg et. al. 2017] T. Greifenberg, S. Hillemacher, B. Rumpe: Towards a Sustainable Artifact Model: Artifacts in Generator-Based Model-Driven Projects. In: Aachener Informatik-Berichte, Software Engineering, Band 30, Shaker Verlag, 2017.

[Jackson 2011] D. Jackson: Software Abstractions: Logic, Language, and Analysis. MIT press, 2011.

[Krcmar et. al. 2014] H. Krcmar, R. Reussner, B. Rumpe: Trusted Cloud Computing. Springer, Switzerland, 2014.

[Lee 2008] Edward A. Lee: Cyber-Physical Systems: Design Challenges. In 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), 2008, pp. 363–369.

[Maoz et. al. 2011] S. Maoz, J. O. Ringert, B. Rumpe: An Operational Semantics for Activity Diagrams using SMV. In: Technical Report. AIB-2011-07, RWTH Aachen University, Aachen, Germany, 2011.

[Müller et. al. 2016] Markus Müller, Klaus Hörmann, Lars Dittmann, Jörg Zimmer: Automotive SPICE in der Praxis: Interpretationshilfe für Anwender und Assessoren. 2edition, dpunkt.verlag, 2016 (available in German only).

[OCL 2014] Object Management Group: Object Constraint Language, 2014. http://www.omg.org/spec/OCL/2.4; accessed on 04/30/2020.

[Roth 2017] Alexander Roth: Adaptable Code Generation of Consistent and Customizable Data-Centric Applications with MontiDex. In: Aachener Informatik-Berichte, Software Engineering: Band 31, Shaker Verlag, 2017.

[Rumpe 2016] B. Rumpe: Modeling with UML: Language, Concepts, Methods. Springer International, 2016.

[Rumpe 2017] B. Rumpe: Agile Modeling with UML: Code Generation, Testing, Refactoring. Springer International, 2017.

[Schindler 2012] M. Schindler: Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P. In: Aachener Informatik-Berichte, Software Engineering. Band 11. Shaker Verlag, 2012 (available in German only).

[SysML 2017] Object Management Group. OMG Systems Modeling Language, 2017. http://www.omg.org/spec/SysML/1.5/; accessed on 04/30/2020.

[UML 2015] Object Management Group. Unified Modeling Language (UML), 2015. http://www.omg.org/spec/UML/; accessed on 04/30/2020.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Jörg Christian Kirchhof, RWTH Aachen University
Michael Nieke, TU Braunschweig
Ina Schaefer, TU Braunschweig
David Schmalzing, RWTH Aachen University
Michael Schulze, pure-systems GmbH

18

Variant and Product Line Co-Evolution

Individual collaborative embedded systems (CESs) in a collaborative system group (CSG) are typically provided by different manufacturers. Variability in such systems is pivotal for deploying a CES in different CSGs and environments. Changing requirements may entail the evolution of a CES. Such changed requirements can be manifold: individual variants of a CES are updated to fix bugs, or the manufacturer changes the entire CES product line to provide new capabilities. Both types of evolution, the variant evolution and the product line evolution, may be performed in parallel. However, neither type of evolution should lead to diverging states of CES variants and the CES product line, otherwise both would be incompatible, it would not be possible to update the CES variants, and it would not be possible to reuse bug fixes of an individual variant for the entire product line. To avoid this divergence, we present an approach for co-evolving variants and product lines, thus ensuring their consistency.

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_18

18.1 Introduction

Product line engineering

Configurability and variability play a pivotal role for collaborative embedded systems (CESs). Individual configurations enable customization and flexibility while, optimally, allowing a high degree of reuse between different *variants*. Product line engineering is an approach that enables mass customization for families of similar (software) systems [Schaefer et al. 2012]. During *domain engineering (DE)*, commonalities and variabilities of variants of a product line—that is, its configured product instances—are typically captured in terms of *features* [Pohl et al. 2005]. A feature represents increments to the functionality of products. *Variability models,* such as feature models [Kang et al. 1990], organize features and the relationships between them. Features are mapped to realization artifacts, such as code, models, or documentation. During *application engineering (AE)*, a variant is derived by defining a configuration that consists of selected features [Pohl et al. 2005]. Using this configuration and the feature-artifact mapping, the resulting artifacts can be composed to form a variant.

Variability for collaborative embedded systems

For collaborative embedded systems (CES), supporting and managing variability is crucial. Typically, a CES is developed once and deployed for different customers and in different environments. Thus, a CES must accommodate customer-specific requirements and be applicable in different environments. Developing these different CES variants individually does not scale economically. Moreover, separate variant development is bad practice as the different variants inevitably diverge from each other, which results in incompatibilities, bugs/errors, and significantly higher maintenance effort [Pohl et al. 2005].

Modifying derived variants

The optimal situation is that all variants are created, maintained, and updated during DE using the product line artifacts and the variability model. In practice, however, customers often require adaptations or updates for their variant, with the adaptations or updates being implemented by changing only this particular variant during AE. For instance, a CES is deployed for one specific customer and this customer requires changes at short notice or implements their own changes. This has several advantages: first, the complexity of implementing such changes is comparably low as the impact on other variants does not have to be considered; second, the time required to deploy new changes and thus the costs are low as well.

www.dbooks.org

This procedure is particularly interesting for variability of CESs. Typically, a CES is used in multiple different CSGs by different companies. Thus, changes to a CES product line require a lot of effort as the impact on all possible variants and the CSGs that use the CES must be considered. Consequently, required changes are implemented directly in a CES variant that is used in a particular CSG.

Diverging changes of product lines and their variants

However, this procedure comes at the cost of lost compatibility between the product line and the changed variant. If product line artifacts are updated, it is unclear whether these changes affect the modified variants and, even worse, it is unclear how to merge the changes at DE level with changes at AE level. As a result, the product line and the modified variants diverge. Consequently, respective variants are not updated if the product line is updated, and other variants cannot benefit from changes that have been made at variant level.

Propagating product line changes to modified variants

To overcome these limitations, we provide an approach that enables engineers to modify variants at AE level while keeping these changes and changes at DE level synchronized. The first part of the approach propagates updates from DE level to modified variants. To this end, an internal repository is automatically maintained. The variants originally derived from the DE level are stored in this repository. If the product line is changed, a three-way-merge mechanism compares the original variant, the updated variant derived from the updated product line, and the modified original variant. As a result, updates from the product line level are merged into the modified variant. Thus, the variant users benefit from product line updates but are still able to modify their variant individually.

Lifting variant changes to product line level

The second part of the approach propagates changes from AE level to DE level. First, changes at variant level are identified. In the next step, the features that are affected by these changes are identified. This is particularly important to allow these changes to be propagated to product line level. However, this task is challenging as, typically, the information about which part of a variant stems from which feature is not preserved when a variant is derived. Finally, the variant changes are transferred semi-automatically to the respective product line at DE level. To this end, regression deltas between original artifacts and modified artifacts are computed and mapped to the respective feature at DE level. As a result, product line artifacts are updated with the most recent changes at the AE level without the need for additional costs to redevelop the variant changes for the entire product line.

18.2 Product Line Engineering

Feature models to represent variability

In product line engineering, features are typically captured in variability models. The most prominent variability model type is a feature model [Batory 2005], [Kang et al. 1990]. Feature models capture the abstract functionality of a product line as features and organize them in a structured tree. Thus, the feature tree has exactly one root feature and can have multiple child features. Each feature, except for the root feature, has exactly one parent feature — that is, the feature tree is an acyclic graph. This tree defines basic relationships between features — that is, a feature can only be selected if its parent feature is selected. Additional constraints can be defined by using feature types or cross-tree constraints in propositional logic with features as variables. In *feature-oriented programming (FOP)*, each feature is implemented separately [Prehofer 1997]. Thus, artifacts, such as code, models, or documentation, that realize a specific feature are developed. In addition, artifacts that are necessary to enable the collaboration of multiple features must be implemented as well.

Variability at implementation level

To realize the variability that artifacts express, there are different mechanisms and notations that establish a feature-artifact mapping. With *annotative* or *negative* approaches, parts of artifacts are marked with feature expressions that define the feature combinations in which they should be used [Schaefer et al. 2012]. If a feature is not selected, its annotated artifact parts are removed. A prominent example of the annotative method is C/C++ preprocessor annotations. With *compositional* or *positive* variability, distinct artifacts for each feature (combination) are implemented that are composed later [Schaefer et al. 2012]. For instance, plug-in systems can be used with a distinct plug-in for each feature. Finally, *transformational* approaches, such as *delta-oriented programming (DOP)* [Clarke et al. 2010], are a combination of the positive and negative approaches. They enable specification of deltas that define changes to artifacts that add, delete, or modify parts of the respective artifacts.

Deriving variants during application engineering

During AE, variants of a product line are derived [Pohl et al. 2005]. To this end, configurations are defined that consist of selected features of the feature model. To derive a concrete variant from such a configuration, a generator uses this configuration, the feature-artifact mapping, and a concrete variability realization mechanism. This variability realization mechanism is specific to the notation used to implement feature artifacts, such as preprocessors, plug-ins, or DOP, and transforms the product line artifacts to match the selected

www.dbooks.org

configuration. For preprocessors, this means removing all annotated parts that do not match the current feature configuration. For additive approaches, such as plug-ins, this means composing all artifacts of the selected features to form a variant. For transformational approaches, such as DOP, the deltas that are mapped to the selected features are collected and their change operations are applied.

Similar to other systems, product lines evolve to meet new requirements or to fix bugs [Schulze et al. 2016]. To this end, feature artifacts and their mapping are modified at DE level and variants can be updated by triggering a new generation at AE level. In theory, this is the optimal way to perform product line evolution. However, in industrial practice, this is often infeasible or simply not done. Consequently, variants are modified at AE level to match specific requirements, to fix bugs, or to be updated. This results in a divergence of product line and variants which we address with the approach presented.

18.3 Propagating Updates from Domain Engineering Level to Application Engineering Level

This section is largely based on [Schulze et al. 2016].

18.3.1 The Challenge of Propagating Updates

Deriving variants and performing customer-specific modifications

To illustrate the process and the resulting problems of propagating updates from DE to AE, we present an abstract overview of variant derivation in conjunction with the evolutionary process described in Figure 18-1. The *Product Line Assets* boxes depicted act as placeholders for different artifacts and each *Variant A* box represents all artifacts belonging to variant A. The creation of a specific customer variant *A* starts with the derivation step at T0, which is symbolized in the figure by Step ①. This step basically consists of multiple actions

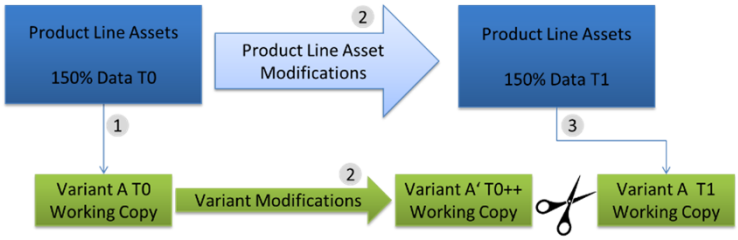


Fig. 18-1: *Challenges of DE and AE co-evolution*

(e.g., selecting features, transforming corresponding artifacts, generating the variant) to be performed for each artifact type, such as requirements, source code, models etc. The result is a working copy for the derived variant that constitutes the base for further development as the product line is not usually able to deliver the entire functionality customers want. Hence, changes to particular artifacts, such as add, remove, and modify, take place on the derived variant at AE level, leading to a customer-adapted and, usually, functionality enriched variant (represented as *Variant A'* in Figure 18-1).

Product line level changes and incompatibilities with variant modifications

Beside modifications on variants' working copies, changes also take place on the entire product line (i.e., DE level) — for example, through maintenance activities such as bug fixing or functionality extension in order to satisfy emerging market needs. The changes at both levels are made simultaneously and in an unsynchronized manner (marked with ② in the figure). In general, this is not a problem and often even desired in industry as it allows variants of different customers to develop at their own speed. However, a problem arises if a derived variant requires further functionality or bug fixes from the product line. This means that the same derivation process of Step ① is performed again at T1 (Step ③), which results in a newly generated working copy for that variant, and as a side effect, all variant modifications (②) on *Variant A* are lost, since the artifacts are replaced by the DE level versions.

The loss of essential changes performed at AE level (visualized by scissors in Figure 18-1) is a major concern for real-world product lines due to the resulting increased time and cost of recreating the changes.

18.3.2 Artifact Evolution and Co-Changes

Basic artifact modifications

Three basic operations can be part of an evolutionary task, regardless of the artifacts affected:

- **Add:** An artifact (e.g., a requirement, code, model, etc.) is added — for example, to extend functionality.
- **Remove:** An artifact is removed — for example, because it became irrelevant.

www.dbooks.org

- **Modify:** An artifact is adapted according to changing circumstances — for example, due to legal issues.[1]

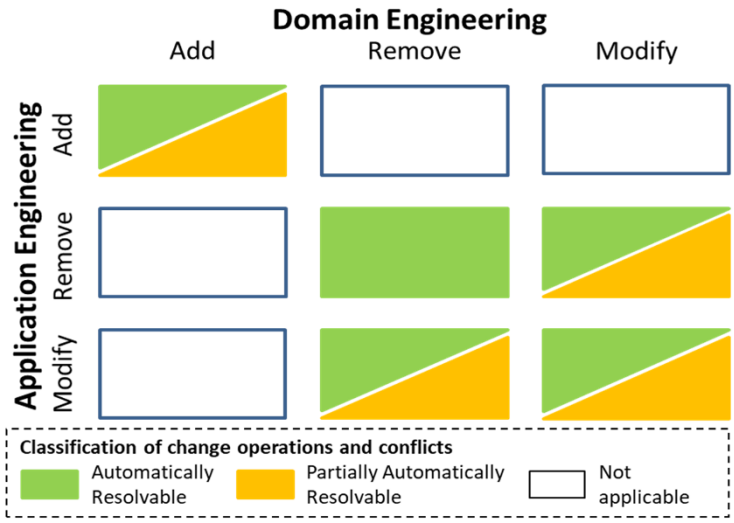


Fig. 18-2: *Co-change operations between DE and AE and their effects*

These types of changes happen at both DE and AE level respectively, and it is only if a change was made on an artifact that exists at both levels that we call it a co-change. Such co-changes can lead to a conflict if an artifact was modified at both levels at the same location but in different ways. In order to preserve the co-changes made at the AE level during update propagation, we have to a) detect, b) classify, and if possible, c) (automatically) resolve each conflicting co-change. The matrix in Figure 18-2 visualizes all possible cases and helps to classify the possible co-changes. As depicted, there are also some cases that can never occur (e.g., an addition of a new artifact at DE level being removed at AE level), other cases that can be fully resolved (e.g., removal of the same artifact at both levels), and cases that can be (partially) automatically resolved (e.g., a modification of a DE level artifact that was removed at AE level). However, before we can classify or even resolve changes, the initial detection of a co-change is key for the subsequent steps.

[1] While this operation can be considered as a combination of the two basic operations add and remove, its semantics is important for determining conflicts. Hence, we treat this operation separately.

Detecting and classifying co-changes

Since an evolution is performed simultaneously at both levels, detecting where a change happened and what type of change it was is essential to enable informed decisions in the subsequent steps. Considering the variants' derivation process in Figure 18-1, a comparison of the artifacts of *Variant A'* with *Variant A* at T1 might be a solution, since a change can easily be detected if an artifact differs between both versions. However, this simple approach is not sufficient to detect the level at which the change happened. More problematically, the most difficult case cannot be uncovered in this way — that is, a case where the same artifact was changed in a different way in both versions. This means that with this two-way comparison, in general, no information about the origin (*Variant A'*, *Variant A* at T1, or both versions) or the kind of change can be retrieved.

The problem of the two-way comparison is that it lacks a common base to compare both variants with. In the derivation process in Figure 18-1, the original working copy *Variant A* at T0 constitutes this common base from which both variants originate. Given this common base, we can use a three-way comparison to obtain the changes between DE and AE. This enables us to compare the evolved variants of DE and AE level not only with each other, but also with their origin — that is, the common base at time T0. As a result, we can determine precisely which change operations were performed on the respective variant. We can therefore classify the changes according to our matrix and thus identify possible conflicts.

Resolving changes

With a full classification for each conflicting co-change, the resolution can be reached partially or full automatically, depending first on the nature of the co-change and second on the resolution strategy — for example, if one level takes precedence during conflict resolution. For most of the cases, this allows a fully automatic resolution. For those cases where conflict resolution needs user assistance, there are often tools that allow for adequate visualization and even merging of the conflict. If such tool support is not available, the user must resolve the conflict by hand, which is in any case the last resort.

18.3.3 Changes to the Variant Derivation Process

Necessity of a common base for three-way comparison

The detection of any possible co-change requires the application of a three-way comparison of the artifacts of three different versions (*Variant A* at T0 and T1, as well as *Variant A'*) of product line variants. However, in the scenario in Figure 18-1, not all the three required

www.dbooks.org

versions are available explicitly. Basically, only *Variant A'* is available and *Variant A* at T1 can be generated from the product line artifacts in their current state. Retrieving the common base version of those two versions is more sophisticated. Generally, two approaches are conceivable to solve this problem as follows.

Regenerating a common base from the product line

In the first approach, the base version is regenerated from the product line, which requires a snapshot of the product line, including generators employed at the point in time when the previous base version was generated (i.e., time T0 in Figure 18-1). Provided that the product line is published in fixed release versions, these snapshots can easily be retrieved even if application engineers have no access to interim versions. However, if there are no such release versions, a snapshot of the entire product line must be created every time a variant generation process is triggered on a changed product line.

Saving generated variants as a common base in a repository

In the second approach, each variant generated is saved in a, possibly local, repository to keep it for later use. This approach is shown in Figure 18-3. Between the DE level and the working copy of a specific variant at AE level, a new level for the repository is introduced that is transparent for application engineers. When application engineers derive a specific variant A for the first time at T0, it is stored automatically in the internal repository for that variant (Step ①). The working copy is initially just cloned from that version (Step ②). Over time, the product line and *Variant A* are changed independently of each other (Step ③). Then, at T1, application engineers want an update of their working copy to synchronize with the current product line version. During that update propagation, a new version of *Variant A* is derived and stored in the internal

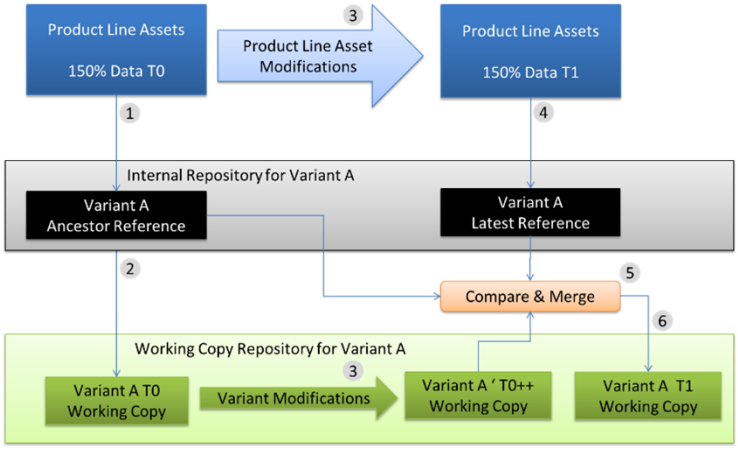


Fig. 18-3: *Solution for co-evolution and propagating updates from DE to AE*

repository (Step ④), but this version is not shown to application engineers directly. Instead, a three-way comparison (Step ⑤) is performed between the two versions in the repository (the ancestor reference as common base and the latest reference) and the working copy version *Variant A'*. As discussed above, most merges are done without user interaction and it is only for conflicts that cannot be resolved that application engineers must decide which changes should be applied. The result is an updated working copy with merged changes of the DE and AE level (Step ⑥). This update process can be repeated each time the product line is changed.

18.3.4 Applicability and Limitations

Basically, our proposed classification scheme is general enough to be applicable with different scenarios and different artifacts in product line development. This is because our definition of both change operations and change conflicts is artifact-independent and we address the integration in the common product line development process. However, due to its general nature, our method requires some manual effort to be adapted for concrete product lines. Most importantly, the concrete artifacts that are subject to change operations must be defined and an instantiation of their granularity levels must be provided. The latter is of specific importance, because the granularity plays a pivotal role in deciding whether a conflict exists or not. Moreover, granularity levels are different for specific artifacts. For instance, for source code, it may be sufficient to distinguish between statement, block, and file level. In contrast, if we consider artifacts in a hierarchical structure, such as requirement specifications, different levels of granularity such as line, section, or subsection may be required to detect conflicts with a suitable accuracy. Finally, developers must specify how the conflict detection and resolution is integrated in the (most likely already existing) development process, for instance, which tools should be used for conflict detection. However, the aforementioned instantiation has to be done only once (when setting up or integrating with an existing product line engineering process) and can subsequently be used for the entire evolution process.

Finally, it is worth mentioning that, with our proposed classification, we focus mainly on syntactical changes. As a result, our classification does not ensure semantic correctness. However, we argue that syntactical correctness is the stepping-stone for consistent

www.dbooks.org

co-evolution in product lines and thus for ensuring integrity of both DE and AE level.

18.3.5 Implementation

In our prototypical implementation, we have integrated the process described into pure::variants[2], the leading industrial variant management tool, which supports the development of product lines. This tool can manage different types of realization artifacts, either by means of generic modeling in the tool or by means of integration into external tools using specific connectors. The derivation process for variants is handled by an extensible set of transformations that are specific to the artifact type or external tool. These transformations are the connection point for our implementation. Since the chosen approach is generic, the prototypical implementation supports all types of artifacts as long as a three-way comparison is available for the specific artifact type. For example, for source code, the internal local repository is realized by simply creating folders for the ancestor as well as latest references, as can be seen in Figure 18-4 from the box in the upper left corner.

The three-way comparison and the merge are then executed using the three directories directly, while specifying the ancestor directory as the common base of the two others once. Thus, when an application engineer wants to update their working copy, they start a new derivation of the current variant, which leads to the generation of a new latest version, followed by triggering the compare and merge operation. If there are no conflicts that have to be resolved manually, the application engineer will get the merged result. If there are conflicts, the application engineer must resolve them by deciding which version—working copy or latest—they prefer to be in the merged result. At the end, the application engineer gets a merged version semi-automatically.

The prototypical implementation was presented to different customers and received a positive response, with many of those customers facing the challenges mentioned with regard to variant and

[2] www.pure-systems.com

product line co-evolution. Thus, our method addresses a highly relevant topic in the industrial domain.

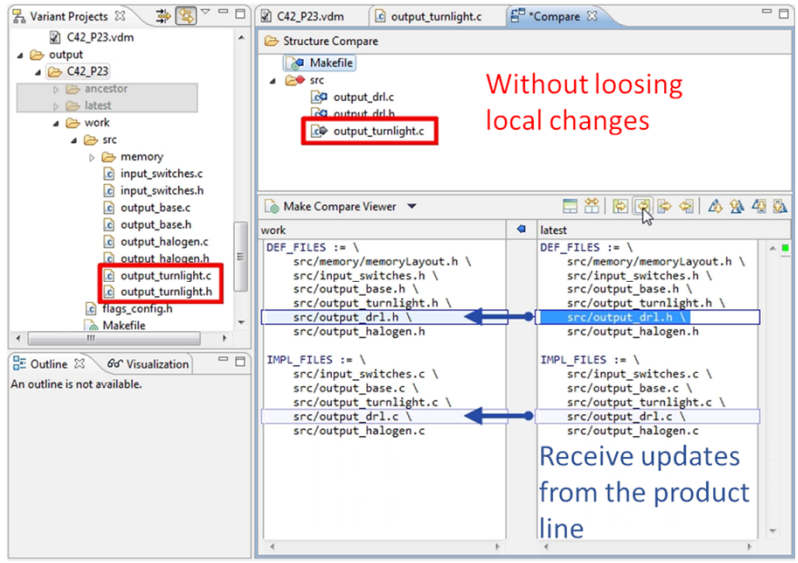


Fig. 18-4: *Updating a variant in pure::variants preserving local changes*

18.4 Propagating Changes from Application Engineering Level to Domain Engineering Level

18.4.1 The Challenge of Lifting Changes

Challenges in propagating changes from the AE level

Propagating updates from the AE level to the DE level produces a few challenges. Introducing changes from the AE level to the DE level may result in conflicts, as development may go ahead at the DE level as well. Detecting changes and applying them to DE level artifacts is made more complicated here, as, in feature-oriented programming, there is often a mapping between features and implementation artifacts. Depending on the variability specification mechanism used, reconstructing the feature mapping from AE level artifacts is often not straightforward. In constructive mechanisms - for example, when constructing a 150% model - references to features may still exist in AE level artifacts. Yet, with transformational approaches, feature references are usually removed during the generation of AE level artifacts. However, reconstructing this mapping on the AE level is crucial for assigning changes to the correct features.

www.dbooks.org

Prerequisites for propagating changes

Our goal is to lower the barrier for adopting changes to variants in product line engineering by supporting the propagation of changes from a variant's working copy to the product line. To adequately propagate changes to the DE level, we have to a) detect changes, b) make the feature information available at AE level, c) assign changes to features or the codebase, and d) resolve each conflicting co-change. We propose a process that detects changes in the working copy of a variant then maps them to the appropriate features and transfers them semi-automatically to the product line.

18.4.2 A Process for Lifting Changes

Approaches to propagating changes

Similar to updating the working copy of variants with changes from the product line, detecting co-changes requires a three-way comparison of the artifacts in questions when lifting changes to the product line. Here, two possible approaches are conceivable. In the first approach, changes in the working copy of the variant (*Variant A'*, see Figure 18-1) are detected by comparing it to its base version (*Variant A at T0*). The changes detected are then translated and applied to the base version of the product line (*Product Line Assets at T0*), resulting in a new product line version. These two versions are then compared with the updated product line (*Product Line Assets at T1*) in a three-way comparison to detect and resolve conflicting co-changes. In the second approach, co-changes are instead detected and resolved on the AE level artifacts and only then translated and applied to the product line. This approach follows the process of updating the working copy of a variant (see Section 18.3.3) with changes from the product line, as co-changes are identified and resolved through a three-way differencing and merge on the three different variant versions.

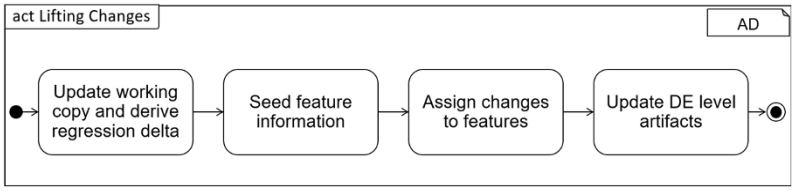


Fig. 18-5: *Activities for propagating changes from the AE level to DE level*

Process description for propagating changes from the AE level

We follow the second approach, as this approach builds upon the previously proposed process for updating a variant. The proposed process for this approach is presented in Figure 18-5. It consists of four steps: first, we update the working copy of a variant with changes

in the product line through a three-way merge on the artifacts of the three different variant versions. In addition to resolving conflicting co-changes, we also calculate regression deltas between the new variant versions (*Variant A* derived from the *Product Line Assets* at T1 and the working copy of *Variant A resulting from the merge*). These regression deltas represent the changes detected that will be applied to the DE level artifacts. However, changes must first be assigned to their corresponding feature (*Seed feature information*), and for this we require access to domain knowledge at AE level. To this end, in the second step, we annotate AE level assets with feature information. These annotations are the input in the third step to assign each change to a corresponding feature. Finally, in the fourth step, we translate and apply changes to DE level artifacts. In the following, we focus on the second and third steps, which we present in more detail.

18.4.3 Deducing Feature Information

Conflicts when updating DE level artifacts

Conflicting co-changes must also be resolved if changes from the AE level are to be propagated to the DE level. Changes must also be assigned to a feature to be made available to other variants of the product line. However, developers at the AE level implement changes concerning the variant's configuration, and information about individual features is usually not available. Changes at AE level can change the implementation of existing features or the codebase (e.g., bug fixing) or add new features (implementation of new functionalities). Before we can assign changes to features, the changes must first be detected, and domain knowledge must be made available at AE level.

Underlying Model

General model description

Artifacts, their content, and their relationships can be represented abstractly as a graph $G = (V, E)$. Here, the set of vertices V represents artifacts or elements of artifacts in the desired granularity, and the set of typed edges $E = V \times V \times T$ represents their relationships, where T is the set of kinds of relationships identified. One possible realization of this data structure is object diagrams, which adequate transformations can extract directly from a development project and which we can employ to identify the impact of individual changes [Butting et al. 2018]. We use this data structure as an internal representation of model artifacts to abstract from concrete syntax changes.

www.dbooks.org

Description of annotations and recommendation values

Besides the internal representation of model artifacts, we annotate elements (vertices) with features, which we store as a mapping $a: V \cup E \rightarrow F$, where F is the set of features. In our representation, the common codebase is mapped to the root feature, which is thereby represented as well. After the second phase (*Seed feature information*), each model element and each relationship of the base variant is annotated with exactly one feature. When assigning changes to features, we calculate recommendation values for each change and feature pair; that is, we calculate a mapping $r : C \times F \rightarrow [0, 1]$ that assigns to each pair *(c, f)* the probability that change c belongs to feature f. Here, $c \in C, and\ f \in F$, where C is the set of changes. Furthermore, we then calculate $r_{rem}(e, f), r_{add}(e, f),\ and\ r_{mod}(e, f)$, which state whether the removal, the addition, or the modification of a model element *e* may belong to a feature *f*.

Seeding Feature Information

Prerequisites for seeding feature information

Since changes in AE level artifacts are applied to model elements of implementation artifacts, information about which model elements belong to which feature is essential to allow informed decisions when assigning changes to features. While feature-oriented programming usually includes a feature mapping that assigns implementation artifacts or even model elements to features, this mapping is usually not available at AE level. The availability of the feature mapping at AE level depends on the variability mechanism and the variant generation process. If feature information is part of implementation artifacts at AE level, then even assigning changes to features may be trivial, as application engineers can implement changes in the scope of the corresponding feature directly.

Required domain knowledge

In most cases, feature information is not part of the resulting implementation artifacts. One example of this is transformational approaches, which transform some core model based on the selected features without traces of these transformations at AE level. As feature information is not available at AE level, we can instead reconstruct this information through the variant generation process. This can either be done directly during the initial variant generation or be recomputed from the product line. With the former, the feature information would have to be computed and derived for all variants, even if changes in a variant are never propagated to the product line. The latter would require the version of the product line, including generators employed at the point in time when the variant was generated. In either case, the goal is to annotate each model element

of the desired granularity of the unmodified variant at AE level with the corresponding feature.

Introducing new artifacts at AE level

In addition to the feature annotation derived from the product line, we require application engineers to annotate which major changes at AE level (e.g., the introduction of new artifacts) represent new features. Since these features are not (yet) known in the product line, it is otherwise not possible to distinguish between new features and changes to an existing feature. In contrast to variability mining, it is not possible to compare several variants to identify new features, since changes usually affect a single working copy of a variant. Instead, by partially annotating changes with a new feature, the full variant may be explored through further analysis. The resulting feature annotation of elements is used in the following to assign changes to specific features.

Assigning Changes to Features

Prerequisites for assigning changes to features

With a complete annotation of the original model elements with features, and incomplete information about new features, we can annotate the remaining changes with features through further analysis. Generally, this can only be achieved partially automatically through a recommendation engine. In some cases, annotating changes with features may be computed fully automatically depending on the quality of analyses employed, the unambiguity of the resulting annotations, and on conflicts in other variants when propagating changes to DE level artifacts.

Noteworthy feature relationships for the recommendation

As before, we focus on the three operations add, remove, and modify. Furthermore, we incorporate domain knowledge into our analysis; that is, we consider the *parent-child* relationship and the *requires* relationship of features. Using well-formedness rules together with domain knowledge enables us to limit the set of features that can contain a particular change. The concrete implementation, however, depends on the modeling language and variability specification mechanism used. The notes here provide the basis for implementing appropriate analyses for the respective circumstances.

Removal of model elements

A model element can only be removed in the feature that introduced it (the annotated feature) or in any of its dependent features. We call a feature f_1 dependent on a feature f_2 if f_1 is in a child-hierarchy of f_2 or if f_1 requires f_2. Dependent features can be removed only if the variability specification mechanisms support removing elements that have been introduced in another feature (e.g., transformational variability specification mechanisms). If model element e is removed at AE level, then $a(e) = f$ (model element e is

www.dbooks.org

annotated with feature f) implies $r_{rem}(rem\ e, f) = 1$ (whether the removal of e may occur in feature f) and in the latter case, this also implies $r_{rem}(rem\ e, f_1) = 1$, where f_1 is dependent on feature f.

Modification of model elements

Similar to the removal of elements, a model element can only be modified in the feature that introduced it or in any of its dependent features. Therefore, if model element e is modified at AE level, then $a(e) = f$ (model element e is annotated with feature f) implies $r_{mod}(mod\ e, f) = 1$ and in the latter case, this also implies $r_{mod}(rem\ e, f_1) = 1$, where f_1 is dependent on feature f.

Addition of model elements

Any domain-specific or general-purpose language supports relationships between model elements, where relationships between two elements can be expressed by the relation $R \subseteq E \times E$, where $(e_1, e_2) \in R$ states that model element e_1 relates to model element e_2 in some way. Common relationships are containment relationships and references to other elements. Examples of the former are classes in Java that contain fields and method declarations. An example of the latter are transitions between two states in an automaton that reference their source and target state. Model elements must be introduced in the same feature that introduces a relationship on that feature, or in any of that feature's parent features - that is, if there is a relationship (e_1, e_2) between model element e_1 and e_2, and $a((e_1, e_2)) = f$ (the relation is annotated with feature f), then $r_{add}(add\ e_1, f) = 1$, $r_{add}(add\ e_2, f) = 1$, $r_{add}(add\ e_1, f_1) = 1$, and $r_{add}(add\ e_2, f_1) = 1$ for all features f_1 in the parent-hierarchy of feature f.

Calculating overall recommendation value

We compute the overall recommendation r for each change with $r(e, f) = r_{rem}(e, f) + r_{add}(e, f) + r_{mod}(e, f)$ by merging the recommendations of r_{rem}, $r_{add,}$ and r_{mod}. The highest recommended feature f for each model element e is returned by the recommendation engine.

18.4.4 Applicability and Limitations

The proposed update process and the proposed recommendation mechanism are general enough to be applicable for different variability specification mechanisms and can be realized for different modeling languages. This is because we generally regard models as constructs consisting of model elements and relationships between these elements. Implementation of the recommendation mechanism and of the update process for different modeling languages, however, requires additional implementation effort, as for each modeling language, we have to identify possible relationships between artifacts

and extract these to transfer them into the recommendation engine. Furthermore, the proposed recommendation mechanism considers all modeling elements and changes to be equally important. If this is not desired, then weights must be defined for these elements. Moreover, domain engineers still have to manually merge changes into the product line artifacts, as recommendations provide only a general idea as to which features particular changes can be applied to. Here, the domain engineers' decisions can be used to limit the decision space further and update recommendations. Updating product line artifacts with changes from the AE level may and will cause conflicts in existing variants. Developers must integrate the process for propagating changes into the product line's development process and define how conflicts across variants will be resolved. Finally, the accuracy of the recommendations depends on the granularity of the overlying model, the maturity of the analysis, and the differencing algorithms employed. Here, we consider only syntactic changes, but algorithms that analyze semantic changes could also be used to enhance recommendations.

18.5 Conclusion

Variability and configurability play a pivotal role for CESs and CSGs. Product line engineering is an approach for structured reuse and management of CES and CSG variability. To meet new requirements, product lines evolve, and their variants can be updated accordingly. However, in industrial practice, individual variants are modified, which yields the threat of incompatibility. In this article, we proposed an approach to keep product lines and their variants synchronized. With this approach, the benefits of performing evolution at both product line level and variant level are combined. With a high degree of automation, engineers can perform evolution at variant level without the drawback of a high manual effort to synchronize the product line with the modified variant. Consequently, our contributions make product line engineering more applicable for industrial practice.

18.6 Literature

[Batory 2005] D. Batory: Feature Models, Grammars, and Propositional Formulas. In: International Conference on Software Product Lines, Springer, Berlin, Heidelberg, September 2005, pp. 7-20.

www.dbooks.org

[Butting et al. 2018] A. Butting, S. Hillemacher, B. Rumpe, D. Schmalzing, A. Wortmann: Shepherding Model Evolution in Model-Driven Development. In: *Modellierung (Workshops), 2018,* pp. 67-77.

[Clarke et al. 2010] D. Clarke, M. Helvensteijn, I. Schaefer: Abstract Delta Modeling. In: Proceedings of the Ninth International Conference on Generative Programming and Component Engineering, ACM, 2010, pp. 13-22.

[Kang et al. 1990] K.C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study (No. CMU/SEI-90-TR-21). Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst., 1990.

[Pohl et al. 2005] K. Pohl, G. Böckle, F. van der Linden: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer 2005, ISBN 978-3-540-24372-4.

[Prehofer 1997] C. Prehofer: Feature-Oriented Programming: A Fresh Look at Objects. In: ECOOP'97 - Object-Oriented Programming, 11th European Conference, Springer, 1997, pp. 419-443.

[Schaefer et al. 2012] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, K. Villela: Software Diversity: State of the Art and Perspectives. In: International Journal on Software Tools for Technology Transfer, Volume 14, Number 5, Springer, 2012, pp. 477-495.

[Schulze et al. 2016] S. Schulze, M. Schulze, U. Ryssel, C. Seidl: Aligning Coevolving Artifacts Between Software Product Lines and Products. In: Proceedings of the Tenth International Workshop on Variability Modelling of Software-Intensive Systems, ACM, 2016, pp. 9-16.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Manfred Broy, Technical University of Munich
Wolfgang Böhm, Technical University of Munich
Bernhard Rumpe, RWTH Aachen University

19

Advanced Systems Engineering

Contribution of the SPES Methodology and Open Research Questions

Advanced systems engineering (ASE) is a new paradigm for agile, efficient, evolutionary, and quality-aware development of complex cyber-physical systems using modern digital technologies and tools. ASE is essentially enabled by smart digital modeling tools for specifying, modeling, testing, simulating, and analyzing the system under development embedded in a coherent and consistent methodology.

The German Federal Ministry of Education and Research (BMBF) projects SPES2020, SPES_XT, and CrESt offer such a methodology and framework for model-based systems engineering (MBSE). The framework provides a comprehensive methodology for MBSE that is independent of tools and modeling languages. The framework also offers a comprehensive set of concrete modeling techniques and activities that build on a formal, mathematical foundation. The SPES framework is based on four principles that are of paramount importance: (1) Functional as well as non-functional requirements fully modeled and understood at system level. (2) Consistent consideration of interfaces at each system level. (3) Decomposition of systems into subsystems and their interfaces. (4) Models for a variety of cross-sectional topics (e.g., variability, safety, dynamics).

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_19

www.dbooks.org

19.1 Introduction

Cyber-physical systems

Many systems and technical products developed today and in the future are or will be cyber-physical systems. These systems exhibit physical as well as smart, complex, and high-performance functionality, are typically not "stand alone," being instead connected to users and to other systems via digital networks such as the Internet, and their services mutually use and complement each other. It is recognizable that to a certain extent, subsystems, which are created by heuristic procedures, are built into the systems, — for example, by "learning procedures."

Typical for those cyber-physical systems is that they embody software intensively, which enables powerful and connected functionalities that go dramatically beyond what was possible in the past for rather isolated mechatronic systems. The high proportion of software leads to an extensive design space in which the most diverse requirements can be identified. Therefore, the identification of a requirements concept is of particular importance. This also creates extensive potential for innovation, both in terms of purely logical functionality but also very much in human-centered human-machine interaction and automation up to full autonomy.

Cyber-physical systems are characterized by the fact that they usually have mechatronic components, especially sensors and actuators to enable the interaction between physical and software components as well as an interaction of the systems with their environment. These new forms of software enable functionalities through the use of advanced software technology, including artificial intelligence methods, and enable human-centered user interfaces for these systems.

Software as a driving factor

It is particularly noteworthy that today's systems contain an extensive proportion of software for good reasons, as this enables functionalities that were completely out of scope even a few years ago. Due to the strong networking, it is obvious to connect systems with completely different tasks and functionalities — in order to use functionality from other systems, but also to make functionality available for other systems and thus increase the degree of automation and optimization.

19.2 Advanced Systems Engineering

System characteristics

The systems of the future are characterized by the following features:

- Extensive software components and functionality, which is mainly determined by the software components
- High degree of networking with other systems for the mutual use of data services
- Strong integration of software with mechanical and electrical components
- Comprehensive, dedicated user interfaces
- High degree of automation up to autonomy
- Continuous further development — including during operation
- Complex integration with business software

Characteristics of the ASE approach

These features are also reflected in the required development approaches and determine the characteristics of the advanced systems engineering (ASE) approach. Accordingly, ASE is characterized by the following features:

- Strong demand for modeling techniques to ensure the correctness of complex functionality and comprehensive tool support
- Frontloading – shifting efforts towards early phases in development
- Strong integration of the development processes of the engineering disciplines involved (mainly software engineering, mechanical engineering, electrical engineering) and the tools used; conventional processes such as sequential, discipline-specific development no longer meet the new requirements
- Strictly systems-centric approach for the holistic integration of the required multidisciplinary design approaches
- Consistency of development across the family of models, with clear semantic foundations, precisely defined relationships between the models, and development steps systematically develop further models from the elaborated model up to the generation of code and test cases based on well-understood semantic coherence.
- Equal support of a top-down and bottom-up approach via the consistency of the transitions between the models.
- Close interlinking of the data-driven and model-driven approach through harmonization of the component and interface concept.
- Intensive use of software tools for all phases of development, consistent artifact orientation, virtual development by creating suitable digital artifacts, automation of the development process

www.dbooks.org

through simulation, generation and automated deduction and quality verification
- Merging of development and operation (continuous development and delivery, DevOps, agility)
- Use of development models for further evolution and during operation (from system model to digital twin)
- New types of cost structures, higher development costs in relation to lower production costs due to the often dramatically higher variability
- Intensive integration of new forms of software and the resulting possibility of adding new and modified functionality even during operation of the systems leads to new types of business models

It is evident that these points interact with, complement, and reinforce each other.

19.3 MBSE as an Essential Basis

Formal system model

The approach pursued by MBSE is clearly distinguished from the document-centered, manual approach that is still widely used today. Objectives, functions, components, interfaces, or quality properties that a system fulfils or provides are described by explicit model elements based on well-defined and well-understood concepts of the domain. A number of modeling concepts are used in model-based development. The concepts are selected in such a way that they capture the essential system properties clearly and precisely. A separate theory can be specified for each of these modeling concepts. The same applies to the description of the relationship between the different modeling concepts used. This has the advantage that users trained in the approach (similar to programming languages) are familiar with the concepts and know which models they have to apply to certain questions. Engineers thus also know the basic problems they have to deal with in order to create the models in a goal-oriented way and use them in system analysis or synthesis. Model-based development is much more than just drawing or setting up models; it also includes the comprehensive use of an elaborated modeling approach.

Pre-built model types, based on a scientific foundation, guarantee properties such as compositionality, which clearly defines the integration of subsystems described by models (such as communication via an interface) and reuse. The models must be coordinated in terms of content and engineers must understand

exactly how the different modeling approaches interact. One important point is the semantic coherence across model boundaries and the boundaries of modeling languages, which ensures that a comprehensive model of the system is created. A system description is then no longer this vaguely informal structure of documents, but rather an interwoven network of standardized models that form a common whole. An instance of the system model, which in this form is then consistently stored in a central model repository ("single point of truth"), is managed. Stakeholders have different views of this central system model that are tailored precisely to their respective roles in the product life cycle (for example, function developer, architect, service). This avoids undetected inconsistencies and, in particular, simplifies the ability to change the models, thus reducing a significant cost driver.

The transition from textual descriptions in natural language to models also has the advantage of reducing ambiguities, making consistency and completeness verifiable, and improving communication between stakeholders. The more formal the model used, the less ambiguity there is in the description. More importantly, formal models enable automatic analyses—for example, to check the interaction of the individual components—and they also allow the use of generators to generate parts of other models and artifacts (such as code or test cases) from elaborated system models.

This shows that model-based development constitutes *one,* perhaps *the* key to advanced systems engineering with a high level of tool support.

Tool-supported development

Another important point here is the possibility of tool-supported development of systems. Here, the software is of particular interest in two respects: on the one hand, the development of systems in their inevitable complexity will be supported in a way that is indispensable to advance such systems in general; on the other hand, supported development requires comprehensive, systematic modeling and thus a virtual registration of the systems. This means that these models can be used as digital twins for the operation of the systems and thus ensure even more extended functionalities. The software optimizes itself during development, so to speak.

Aspects of MBSE

Three aspects of MBSE must be considered separately, but nevertheless skillfully coordinated with each other:

Methodology: A system model, which in turn consists of a multitude of model types, is itself a complex artifact that cannot be created effectively and efficiently without an underlying science-based methodology. Therefore, an MBSE methodology includes the

www.dbooks.org

definition of the relevant model types and their relationships. Furthermore, it defines views of the system model, which structure the complex overall model into several, less complex models that are adapted to the given development situation. Examples of views include functional and logical or technical architecture views. An MBSE methodology also describes possibilities for analysis and generation of the specific models. The degree of formalization of the models defined in the system model determines the degree of automation of analyses and model generation. A high degree of automation, which of course must also be supported by the tools used, allows in turn an iterative and agile development process, such as in pure software development.

Modeling language: The modeling language defines the syntax and semantics used to describe the models of the methodology in concrete terms — for example, which textual or graphical notations are allowed (syntax). The semantics of a modeling language defines the meaning of these notations. The problem here is that many of the common modeling languages (such as SysML) have at best a loosely defined semantics. This causes problems similar to those of natural language descriptions.

Tools: The methodology and language must be supported by appropriate tools in order to make efficient use of the possibilities offered by models. It is also crucial that the tool chains used are compatible with each other and that the tools used support the chosen methodology and the modeling language both syntactically and semantically and with a high degree of automation.

19.4 The Integrated Approach of SPES and SPES_XT

Principles of the SPES modeling framework

In the BMBF project SPES2020 [Pohl et al. 2012] and its successors SPES_XT [Pohl et al. 2016] and CrESt, a methodology and framework for MBSE were developed that allow efficient model-based development of embedded systems. The SPES framework provides a comprehensive methodology for MBSE that is independent of tools and modeling languages. The framework also offers a comprehensive set of concrete modeling techniques and activities that build on a formal, mathematical foundation. The SPES framework is based on four principles of paramount importance:

- Functional as well as non-functional requirements fully modeled at system level using appropriate abstractions (views)
- Consistent consideration of interfaces at each level

- Decomposition of the interface behavior and the description of systems via subsystems and components at different levels of granularity
- Definition of models based on the above principles for a variety of cross-sectional topics (variability, safety, etc.) and analysis options

System model in SPES

A system model in the SPES approach is a conceptual ("generic") model for the description of systems and their properties, consisting of:

- Models for the operational context that influences or is influenced by the system at runtime
- Models of the interface that clearly delimit the system from its operational context
- A behavior of the system that can be observed at the interface
- Models of the internal structure of the system implemented by state machines or by interrelated and communicating subsystems (architecture) to which the SPES framework can be recursively applied

Universal interface concept

The core of the methodology is the *universal interface concept*, which defines interfaces for all elements, each consisting of the interface syntax and a description of the behavior observable at the element boundary. Requirements, functions, and logical or technical components are thus described via the interface and are connected to each other via their interfaces. The interface concept provides the basic decomposition and modularity.

Views and crosscutting topics

Views [IEEE42010 2011] in the SPES framework are the requirements view, functional view, logical view, and technical view. They decompose the system into the logical or technical components involved. Crosscutting topics supplement the models of the views accordingly. For example, this allows aspects of the functional safety of systems to be described and analyzed. The SPES framework is open to the addition of new views, such as a geometric view.

Levels of granularity

In order to make the complexity of the system and the associated development process manageable, relevant architectural components are considered as independent (sub)systems according to the principle of "divide and rule." For these systems, models and views are created according to the SPES approach. This creates predefined views for the system and its subsystems with matching levels of granularity. The modeling of the system at the different levels of granularity determines the subject of the discourse (scope) and is an

www.dbooks.org

important tool in model-based development to reduce system complexity and to make the development process manageable.

Mathematical foundation

The SPES MBSE methodology follows a strict system-centric approach that specifies a system at several levels of granularity. At the highest level of granularity, there are always the models that represent the system under consideration as a whole. At (varying numbers of) further levels of granularity, increasingly fine subsystems are successively considered, and further details are modeled. Although "top-down" is the basic principle, iterative, agile, and evolutionary processes are also supported. The mathematical model FOCUS, on which the SPES framework is based, ensures the consistency of the models of systems and subsystems. Levels of granularity help to (1) control the complexity of the system under consideration, (2) perform checks on the system at different levels of complexity, (3) distribute development tasks—for example, to suppliers—and (4) reuse individual models several times. Since the principle of granularity levels is based only on the interface concept, the mechanism allows the integration of the different engineering disciplines (mechanical engineering, electrical engineering, software engineering). As long as the interface concept is realized, the methods, processes, or tools used to develop the subsystems at the lower levels of granularity are irrelevant.

Consistency of the models

Besides abstraction and granularity, consistency is an important feature of the models in the SPES framework. We distinguish between horizontal consistency and vertical consistency. Two models are horizontally consistent if they belong to different views of the same system (i.e., are within one level of granularity) and do not represent contradictory properties of the system under consideration. Two models are vertically consistent if they belong to one view at different levels of granularity and do not represent contradictory properties of the system under consideration with regard to the specific view.

Agile and iterative development

The SPES framework does not specify the order in which the different models should be created for the views. Thus, the SPES method can be used to implement top-down as well as bottom-up approaches and iterative or incremental development and even evolution. As mentioned above, the mechanism of granularity levels allows the integration of different approaches and development tools, as typically required for mechatronic systems. Since the formal basis of the SPES methodology also supports under-specification, it is possible to extend and successively refine the models iteratively step by step. This means in particular that the model-based approach does not contradict but rather supplements the basic principles of agile

development. Techniques such as "continuous integration" can also be used in a purposeful manner. It should be emphasized that this form of an agile approach is not just code-centric but also model-centric.

Extension towards networks of systems

In the CrESt project, the SPES framework was extended to support collaboration and dynamics (formation of system networks at runtime) in systems. The existing viewpoint structure was essentially retained, but the models contained within the structure were extended by additional model types and information.

19.5 Methodological Extensions: From SPES to ASE

SPES contribution to ASE

Advanced systems engineering (ASE) is definitely a new paradigm for agile, efficient, evolutionary, and quality-aware development of complex cyber-physical systems using modern digital technologies and tools. As said earlier, ASE is essentially enabled by smart digital modeling tools for specifying, modeling, testing, simulating, and analyzing the system under development embedded in a coherent and consistent methodology.

Model-based systems engineering is thus a core element of ASE and the SPES methodology, as a fully model-based approach, therefore provides an excellent basis for ASE. In particular, the SPES methodology includes:

- Consistent models that cover the entire product development process
- A variety of modeling techniques to ensure and analyze the correctness of complex functionality
- Modularity and decomposition, which allow reuse of model elements at all levels
- Consistent architecture views and executable model elements, which allow functional prototypes and automated analyses in early phases of the development process (frontloading)
- Strict system-centric approach to support the necessary multidisciplinary design approaches
- Integration of the development processes of the engineering disciplines involved (computer science, mechanical engineering, electrical engineering) and the tools used there via the concept of granularity levels
- Extensive models especially for software engineering and strong integration of software with mechanical and electrical components

www.dbooks.org

- Extension of the SPES framework towards aspects such as dynamic networking and collaboration of systems at runtime in the CrESt project; for this purpose, a number of additional crosscutting topics were defined, and the models of the existing viewpoints were supplemented accordingly

New methodological and crosscutting issues would be, for example:

- Extension of the predominantly discrete models to analog models; integration of control engineering approaches — keyword "interdisciplinary modeling"
- Integration of novel methods for the generation of subsystems and their behavior through big data and machine learning
- Integration of security models for safety and security into model-driven development with a focus on certification
- Consideration of digital twins as part of the overall system to be developed
- Quality assurance at runtime
- System qualification and certification
- Dedicated user interfaces

Further development towards ASE

Up to now, the development of the SPES framework has focused exclusively on the product development process. At the same time, SPES offers the possibility to add new viewpoints to the already existing viewpoints or to extend the existing viewpoints via additional crosscutting topics and integrate them into the framework. A further development towards ASE should therefore take into account extensions towards the entire product life cycle, including models and extensions for market and business models as well as system operation and service models.

SysML as a modeling language

The models, methodology, and techniques developed in the SPES, SPES_XT, and CrESt projects were deliberately written independently of a specific modeling language in order to ensure the greatest possible range of application. In industrial practice, especially in small and medium-sized enterprises, it has been shown that almost all MBSE implementation projects rely on SysML as a modeling language, despite all the open questions and shortcomings associated with it. The reason for this is the spread of SysML in companies as well as the support of SysML in many MBSE tools available in practice. Due to the spread and acceptance of SysML, it must be explicitly supported as a modeling language both syntactically and semantically with the SPES methodology. A research project based on the SPES framework is planned that will break down the current barriers to the industrial introduction of MBSE and thus pave the way for a broad industrial

adoption of the SPES methodology based on common language syntax and pragmatic tools.

Artificial intelligence

Parts of future systems will be determined by the use of techniques such as machine learning (ML) or, more generally, artificial intelligence (AI). Integrating AI components into embedded systems leverages the considerable potential of current and future AI technologies in embedded systems. Their use enables future embedded systems to suitably process the constantly growing volume of information resulting from digitalization and to adapt to changing conditions and to the knowledge gained from the data at runtime. In order to be able to develop such systems efficiently, the explicit modeling methods available must be extended by implicitly learned modeling techniques. In principle, the approach presented here is already suitable for systems that have AI components. The universal interface concept of the SPES framework provides a sustainable basis for this. However, it has to be considered that the behavior of such systems is subject to a certain variability during runtime — for example, if the component continues to learn during runtime.

Quality properties

One central challenge for the integration of AI methods into embedded systems is therefore the guarantee (verifiability) of the essential functionality and quality properties of the systems — and this despite the fact that system components cannot be completely specified and are often non-deterministic or even dynamically adapting due to adaptations of the systems at runtime that could not be foreseen at development time.

19.6 Conclusion

ASE requires a clean scientific foundation and a consistent integration of software development and system development methods when designing software-intensive cyber-physical systems. Central to advanced systems engineering is the use of digital techniques in both the product and the development process and the exploitation of the synergies between them. The preliminary work in the area of model-based development of software-intensive systems offers an ideal entry point. Nothing less than a paradigm shift from the engineering of mechanical machines to the integrated engineering of networked, information-centric mechanical systems must be mastered.

www.dbooks.org

19.7 Literature

[Broy 2010] M. Broy: A Logical Basis for Component-Oriented Software and Systems Engineering. In: The Computer Journal, Vol. 53, No. 10, 2010, pp. 1758-1782.

[Broy and Rumpe 2007] M. Broy, B. Rumpe: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. In: Informatik-Spektrum. Springer Verlag, Band 30, Heft 1, 2007 (available in German only).

[Broy and Stølen 2001] M. Broy, K. Stølen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement, Springer, 2001.

[Broy et al. 2007] M. Broy, M. L. Crane, J. Dingel, A. Hartmann, B. Rumpe, B. Selic: UML 2 Semantics Symposium: Formal Semantics for UML. In: Models in Software Engineering. Workshops and Symposia at Models 2006. Genoa, LNCS 4364, Springer, 2007.

[Broy et al. 2020] M. Broy, W. Böhm, M. Junker, A. Vogelsang, S. Voss: Praxisnahe Einführung von MBSE – Vorgehen und Lessons Learnt, White Paper, fortiss GmbH, 2020 (available in German only).

[IEEE42010 2011] ISO/IEC/IEEE 42010:2011: Systems and Software Engineering — Architecture Description. International Organization for Standardization, 2011.

[Pohl et al. 2012] K. Pohl, H. Hönninger, R. Achatz, M. Broy: Model-Based Engineering of Embedded Systems, Springer, 2012.

[Pohl et al. 2016] K. Pohl, M. Broy, M. Daembkes, H. Hönninger: Advanced Model-Based Engineering of Embedded Systems, Extensions of the SPES 2020 Methodology, Springer, 2016.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Appendices

www.dbooks.org

A – Author Index

© The Author(s) 2021
W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0

A

Akili, Samira
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany

Albers, Dr. Karsten
INCHRON AG
Neumühle 24-26
91056 Erlangen, Germany

Aluko Obe, Patricia
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

B

Bandyszak, Torsten
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

Böhm, Birthe
Siemens AG
Corporate Technology
Günther-Scharowsky-Str. 1
91058 Erlangen, Germany

Böhm, Dr. Wolfgang
Technical University of Munich (TUM)
Department of Informatics
Boltzmannstr. 3
85748 Garching, Germany

Bolte, Dr. Benjamin
itemis AG
Am Brambusch 15
44536 Lünen, Germany

Brings, Jennifer
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

Broy, Prof. Dr. Dr. h.c. Manfred
Technical University of Munich (TUM)
Department of Informatics
Boltzmannstr. 3
85748 Garching, Germany

Butting, Arvid
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

C

Caesar, Birte
Helmut Schmidt University Hamburg
Institute of Automation Technology
Holstenhofweg 85
22043 Hamburg, Germany

Cârlan, Carmen
fortiss GmbH
Software & Systems Engineering
Guerickestr. 25
80805 Munich, Germany

Cioroaica, Emilia
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

www.dbooks.org

D

Daun, Dr. Marian
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

Dimitrov, Dimitar
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

F

Fay, Prof. Dr.-Ing. Alexander
Helmut Schmidt University Hamburg
Institute of Automation Technology
Holstenhofweg 85
22043 Hamburg, Germany

Feeken, Linda
Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany

G

Gandor, Malin
Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany

Gerostathopoulos, Dr. Ilias
Technical University of Munich (TUM)
Boltzmannstr. 3
85748 Garching, Germany

Granrath, Christian
RWTH Aachen University
Junior professorship for mechatronic systems for combustion engines
Forckenbeckstr. 4
52074 Aachen, Germany

H

Hayward, Alexander
Helmut Schmidt University Hamburg
Institute of Automation Technology
Holstenhofweg 85
22043 Hamburg, Germany

Hildebrandt, Constantin
Helmut Schmidt University Hamburg
Institute of Automation Technology
Holstenhofweg 85
22043 Hamburg, Germany

Hillemacher, Steffen
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

J

Jäckel, Dr. Nicolas
FEV Europe GmbH
Ingolstädter Str. 49
80807 Munich, Germany

Jöckel, Lisa
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

K

Käser, Lorenz
PikeTec GmbH
Waldenserstr. 2-4
10551 Berlin, Germany

Kirchhof, Jörg Christian
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

Kläs, Michael
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

Klein, Dr. Cornel
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Klein, Dr. Wolfram
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Koo, Chee Hung
Robert Bosch GmbH
Corporate Sector Research and Advance Engineering
Robert-Bosch-Campus 1
71272 Renningen, Germany

Krajinski, Lisa
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

Kranz, Sieglinde
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Kugler, Christopher
FEV Europe GmbH
Neuenhofstr. 181
52078 Aachen, Germany

Kuhn, Dr. Thomas
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

L

Laxman, Nishanth
Technical University of Kaiserslautern
Department of Computer Science
Gottlieb-Daimler-Str. 47
67653 Kaiserslautern, Germany

M

Malik, Dr. Vincent
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

www.dbooks.org

Marmsoler, Dr. Diego
Technical University of Munich (TUM)
Department of Informatics
Boltzmannstr. 3
85748 Garching, Germany

Meyer, Max-Arno
RWTH Aachen University
Junior professorship for mechatronic systems for combustion engines
Forckenbeckstr. 4
52074 Aachen, Germany

Mirzaei, Elham
InSystems Automation GmbH
Wagner-Régeny-Str. 16
12489 Berlin, Germany

Möhrle, Felix
Technical University of Kaiserslautern
Department of Computer Science
Gottlieb-Daimler-Str. 47
67663 Kaiserslautern, Germany

N

Neumann, Martin
InSystems Automation GmbH
Wagner-Régeny-Str. 16
12489 Berlin, Germany

Nieke, Michael
Technische Universität Braunschweig
Institute of Software Engineering and Automotive Informatics
Mühlenpfordtstr. 23
38106 Braunschweig, Germany

Nickles, Jochen
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

O

Orth, Dr. Philipp
FEV Europe GmbH
Neuenhofstr. 181
52078 Aachen, Germany

P

Petrovska, Ana
Technical University Munich (TUM)
Department of Informatics
Boltzmannstr. 3
85748 Garching, Germany

Pohl, Prof. Dr. Klaus
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

Pudlitz, Florian
Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany

R

Regnat, Nikolaus
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Rösel, Simon
Model Engineering Solutions GmbH
Waldenserstr. 2-4
10551 Berlin, Germany

Rosen, Roland
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Rothbauer, Stefan
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Rumpe, Prof. Dr. Bernhard
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

S

Safdari, Samira
Expleo Germany GmbH
Wilhelm-Wagenfeld-Str. 1-3
80807 Munich, Germany

Sauer, Dr. Markus
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Schaefer, Prof. Dr. Ina
Technische Universität Braunschweig
Institute of Software Engineering and
Automotive Informatics
Mühlenpfordtstr. 23
38106 Braunschweig, Germany

Schlie, Alexander
Technische Universität Braunschweig
Institute of Software Engineering and
Automotive Informatics
Mühlenpfordtstr. 23
38106 Braunschweig, Germany

Schlingloff, Prof. Dr. Holger
Fraunhofer Institute for Open
Communication Systems FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany

Schmalzing, David
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

Schneider, Dr.-Ing. Daniel
Fraunhofer Institute for Experimental
Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

Schröck, Dr.-Ing. Sebastian
Robert Bosch GmbH
Corporate Sector Research and Advance
Engineering
Robert-Bosch-Campus 1
71272 Renningen, Germany

Schulze, Dr. Michael
pure-systems GmbH
Otto-von-Guericke-Str. 28
39104 Magdeburg, Germany

Sohr, Dr. Annelie
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Stierand, Dr. Ingo
Oldenburg Institute for Information
Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany

www.dbooks.org

Straße, Dr. Alexander auf der
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

T

Terfloth, Axel
itemis AG
Am Brambusch 15
44536 Lünen, Germany

Toborg, Steffen
PikeTec GmbH
Waldenserstr. 2-4
10551 Berlin, Germany

Törsleff, Sebastian
Helmut Schmidt University Hamburg
Institute of Automation Technology
Holstenhofweg 85
22043 Hamburg, Germany

U

Unverdorben, Stephan
Siemens AG
Corporate Technology
Günther-Scharowsky-Str. 1
91058 Erlangen, Germany

V

Velasco Moncada, David Santiago
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

Vogelsang, Prof. Dr. Andreas
Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany

Vollmar, Jan
Siemens AG
Corporate Technology
Günther-Scharowsky-Str. 1
91058 Erlangen, Germany

Voss, Dr. Sebastian
fortiss GmbH
Software & Systems Engineering
Guerickestr. 25
80805 Munich, Germany

W

Wachtmeister, Louis
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

Wehrstedt, Dr. Jan Christoph
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Weyer, Dr. Thorsten
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen, Germany

Wirtz, Boris
Oldenburg Institute for Information Technology (OFFIS)
Escherweg 2
26121 Oldenburg, Germany

Wißdorf, Anna
PikeTec GmbH
Waldenserstr. 2-4
10551 Berlin, Germany

Wolf, Stefanie
Siemens AG
Corporate Technology
Günther-Scharowsky-Str. 1
91058 Erlangen, Germany

Wortmann, Dr. Andreas
RWTH Aachen University
Software Engineering
Ahornstr. 55
52074 Aachen, Germany

Z

Zeller, Dr. Marc
Siemens AG
Corporate Technology
Otto-Hahn-Ring 6
81739 Munich, Germany

Zernickel, Jan-Stefan
InSystems Automation GmbH
Wagner-Régeny-Str. 16
12489 Berlin, Germany

www.dbooks.org

B – Partner

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0

Bertrandt GmbH

The Bertrandt Group has been providing development solutions for the international automotive and aviation industry for over 35 years. A total of around 11,000 employees at 44 locations stand for in-depth know-how, future-oriented project solutions and a high degree of customer orientation.

In the dynamic environment of the automotive and aviation industry, the complexity of individual mobility solutions is constantly increasing. The trends towards environmentally friendly mobility, networking, safety and comfort today require comprehensive technical know-how in product development. As a co-designer of sustainable mobility, Bertrandt is constantly adapting its range of services to the needs of its customers and to changing market conditions. For the international automotive industry, the range of services covers the entire value-added chain of product creation: from the initial idea, through the development and validation of components, modules and systems, to complete vehicles with related services such as quality, supplier and project management or training.

In the field of electronics development, Bertrandt's activities range from the classic areas (infotainment, comfort, chassis, on-board networks, etc.) to the current and new challenges surrounding electrified driving and vehicle networking (Car2X) in the areas of driver assistance systems, automated driving, online services/apps, infrastructure/IT. At the Ingolstadt site, among others, holistic solutions for the automotive industry are developed with a focus on bodywork, interior, electrics/electronics with its own electronics center, powertrain, FE simulation/calculation and testing/trial. In the field of driver assistance systems, the entire development process is covered here, from requirements analysis to software development and overall system testing in various projects.

→ www.bertrandt.com

Expleo Germany GmbH

The Expleo Group is a leading international engineering partner with over 12,500 employees in 20 countries worldwide. In Germany, Expleo Germany GmbH with around 1,100 employees offers engineering and product solutions for the automotive, aerospace, industry and transportation sectors - together with its subsidiaries SILVER ATENA and Automotive Solutions Germany (ASG). At 15 locations Expleo develops, tests and supplies software, electronics, special mechanical solutions and customer-specific lighting systems - from the technological idea to series production. In the automotive sector Expleo designs highly automated, networked and electrified mobility and offers demand-based products, components and tools. Expleo bundles expertise and their own IP in specialized Competence Centers.

→ www.expleo-germany.com

www.dbooks.org

FEV Europe GmbH

FEV is a leading independent international service provider of vehicle and powertrain development for hardware and software. The range of competencies includes the development and testing of innovative solutions up to series production and all related consulting services. The range of services for vehicle development includes the design of body and chassis, including the fine tuning of overall vehicle attributes such as driving behavior and NVH. FEV also develops innovative lighting systems and solutions for autonomous driving and connectivity. The electrification activities of powertrains cover powerful battery systems, e-machines and inverters. Additionally, FEV develops highly efficient gasoline and diesel engines, transmissions, EDUs as well as fuel cell systems and facilitates their integration into vehicles suitable for homologation. Alternative fuels are a further area of development.

The service portfolio is completed by tailor-made test benches and measurement technology, as well as software solutions that allow efficient transfer of the essential development steps of the above-mentioned developments, from the road to the test bench or simulation.

The FEV Group is growing continuously and currently employs 6700 highly qualified specialists in customer-oriented development centers at more than 40 locations on five continents.

→ www.fev.com

fortiss GmbH

The Munich research and transfer institute for software-intensive systems, fortiss GmbH, was founded in 2009 as an affiliated institute of the TU Munich together with the Fraunhofer Gesellschaft and the LfA Förderbank Bayern. The Department of Software and Systems Engineering (Head of Department PD Dr. Schätz) is involved in the project. The department develops cross-domain integrated software and system architectures and related development methods and tools. The department offers comprehensive competence in modern methods and tools for the professional development of software-intensive systems, starting with the elicitation of requirements up to verification and integration. In particular, the goal is to prepare and apply basic methods - such as formal model checking or automatic design space exploration - for engineering applications on an industrial scale. The main focus is on automotive, avionics, rail and energy systems, among others.

→ www.fortiss.org

Fraunhofer Institute for Open Communication Systems FOKUS

The Fraunhofer Society for the Advancement of Applied Research is the biggest organization for applied research and development services in Europe, and FOKUS is the largest Fraunhofer institute in the field of Information and Communications Technology. Its main topic is digital networking and its effects on society, economy and technology. Since 1988 it has been supporting commercial enterprises and public administration in the design and implementation of digital change. To this end, Fraunhofer FOKUS offers research services ranging from requirements analysis, consulting, feasibility studies, technology development to prototypes and pilots in its business units. The system quality center of FOKUS is specialized in quality engineering for the internet of things. Via its chief scientist, Prof. Schlingloff, it has strong academic foundations and close connections to Humboldt Universität. It offers services in model-based development and testing of software-based systems, tool development and tool integration, test design and automation, and support of product qualification and certification. The group provides methods, processes and tools for the development and quality assurance of software-intense systems and services.

→ www.fokus.fraunhofer.de

Fraunhofer Institute for Experimental Software Engineering (IESE)

The Fraunhofer Institute for Experimental Software Engineering (IESE) was founded in 1996 and is one of 60 institutes of the internationally operating Fraunhofer-Gesellschaft. IESE currently employs more than 200 people, whose goal is to sustainably transfer scientific results into industrial applications through applied research. One focus of Fraunhofer IESE's work is on methods for developing highly reliable and safety-critical software-intensive embedded systems. IESE's budget volume is well over 12 million euros, and is largely derived from industrial contract research and collaborative and research projects involving industry. Over the past years, Fraunhofer IESE has been involved in a large number of research projects in various fields, such as functional safety, Big Data, or processes, and in many cases has taken leading roles in the project. At the same time, there have been and still are numerous industrial projects thematically related to CrESt, covering a large number of application domains (automotive, agricultural engineering, medical technology, defense technology, aerospace, mining, railway engineering). IESE has already been significantly involved in the research projects SPES 2020 and SPES_XT, in particular with contributions to modular model-based safety cases as well as the combination of heterogeneous safety analyses.

→ www.iese.fraunhofer.de

www.dbooks.org

Helmut Schmidt University Hamburg

Under the guidance of Prof. Dr.-Ing. Alexander Fay, research at the Institute of Automation Technology at Helmut Schmidt University Hamburg has been focused on modelling languages, methods, and tools for the efficient engineering of complex automation systems, e.g. in manufacturing, process industry, transportation, buildings, and energy distribution. A key element of our research is the creation and use of information models throughout the lifecycle of these systems. These information models are developed and applied for the design, implementation, testing, operation, and modernization of existing systems. A hot topic is how to increase flexibility in such complex systems with the help of modularity and system collaboration, which entails the need to deal with incomplete and inconsistent information models. The institute collaborates with major automation suppliers as well as operators of automated facilities to implement research results and to tackle new research problems of practical importance.

→ www.hsu-hh.de

Humboldt-Universität zu Berlin

Humboldt-Universität zu Berlin was the first German university to introduce the unity of research and teaching, to uphold the ideal of research without restrictions and to provide a comprehensive education for its students. Today, Humboldt-Universität is in all rankings among the top German universities. It was chosen “University of Excellence” in June 2012, with a renewed labelling within the Berlin University Alliance in 2019. The computer science department of Humboldt-Universität was founded in 1989. It encompasses 21 research groups, structured into the three clusters "Data and Knowledge Engineering", "Algorithms and Structures", and "Model-driven systems engineering". The research group "Specification, Verification and Test Theory" is headed by Prof. Schlingloff. The group has been working for 15 years on formal methods of software development, mainly in the field of safety-critical embedded systems. It has close connections to Fraunhofer FOKUS, where Prof. Schlingloff is chief scientist. Current research topics include quality assurance of embedded control software, model-based development and model checking, logical specification and verification of requirements, automated software testing, and online monitoring of safety-critical systems with formal methods.

→ www.hu-berlin.de

INCHRON AG

INCHRON AG is a specialist in the development methodology of embedded systems with hard real-time requirements. Our mission is to support our customers with our knowledge, experience, advanced tools, and broad industry expertise in the development of embedded systems of any kind and complexity.

With our sophisticated methodology, which undergoes continuous refinement, we shape the future of embedded systems development. The INCHRON Tool-Suite is an essential part of our methodology and provides state-of-the-art tools for analysis, simulation, optimization, and detailed prediction of the dynamic behavior of embedded software. Its successful practical use and integration into development processes of varying operational domains serve to prove the outstanding capabilities of this unique tool. Areas of expertise include:

- Detailed analysis of the performance and runtime behavior of embedded systems of any complexity using simulation and worst-case analysis.
- Automated optimization of the dynamic behavior of stand-alone or distributed systems.
- Design and early analysis of new/changed system architectures through frontloading.
- Efficient porting of single-core software to multi-core processors.
- Adaptation of existing software to alternative networking technologies, such as FlexRay or Ethernet.
- End-to-end timing analysis of event chains, from sensor to actuator, via ECUs or Domain Controllers and in-vehicle networks.
- Detailed documentation of real-time requirements and their degree of conformance.
- Determination and elimination of the causes of runtime errors such as interrupt and task displacement, life/deadlocks of tasks, or stack overflows.
- Detailed analysis of complex scheduling scenarios.
- Trace analysis and trace visualization (Lauterbach, iSYSTEM, and other proprietary formats).
- Support for industry-specific standards such as AUTOSAR, ARINC-653, AFDX.
- Functional safety (ISO 26262).
- Increased level of test coverage through statistical analysis of compliance with real-time requirements, combined with stress tests and robustness analyses.

Autonomous driving is a key focus application for INCHRON. Our customers are already using our approach and tools with great success in the design, optimization, and testing of modern driver assistance systems (ADAS), the preliminary stage to autonomous driving. The use of the solutions INCHRON provides will prove indispensable in the future in coping with the exceptional complexity of such advanced automotive platforms.

→ www.inchron.com

www.dbooks.org

InSystems Automation GmbH

InSystems Automation develops innovative automation technology and special machines for production, material flow and quality control. The range of services covers all tasks from the creation of specifications, electrical project planning, installation and programming to commissioning, maintenance and service.

Customers include large and medium-sized manufacturing companies from the cosmetics, pharmaceutical, printing and automotive industries. Compared to competitors, InSystems distinguishes itself primarily by the holistic approach: Construction, mechanical engineering, conveyor technology and software are completely created in-house at InSystems. The company was founded in 1999 by the two managing directors Henry Stubert and Torsten Gast and has grown steadily since then. In the meantime, the company has more than 50 employees and is located in the scientific location Berlin-Adlershof. Further subsidiaries are the independent InSystems Vertriebsgesellschaft mbH in Fürth and InSystems Automation, Inc. in Washington, North Carolina USA. Since 2012, InSystems has specialized in the production of autonomously navigating transport robots developed under the brand name proANT. These robots are manufactured for loads from 30 to 1,000 kg depending on the customer's requirements and are implemented as a fleet into an existing production control system. The vehicles navigate automatically using laser scanners and react independently to changes in their working environment. The vehicles are designed for personal safety and can work with workers without the need for additional safety precautions such as safety fences or separation of traffic routes. If an obstacle appears in the safety field, the vehicle reduces its speed, navigates around the obstacle or stops.

Using a stored environment map, each vehicle independently calculates the optimum route to the destination. The vehicles can be integrated into the production process individually or as a fleet. The vehicles communicate with each other via an encrypted WLAN and avoid each other at an early stage. This prevents traffic jams or mutual obstruction. In addition, the battery condition of the vehicles is regularly checked by a fleet manager, who sends them to the charging station when the charge level is low. In the Showroom Industrie 4.0 (Wagner-Régeny-Str. 16, 12489 Berlin) visitors can get a live impression of the driving behaviour of the transport robots, how they cleverly avoid people and obstacles, automatic load transfers and the supply of a manual workstation.

→ www.asti-insystems.de

itemis AG

itemis AG is a specialist for model-based software and systems engineering and integrated, modular tool chains. Itemis AG is a leader in developing domain-specific modelling environments on the open source platform Eclipse. With 200 employees, itemis works in Germany and with branches in France, Switzerland and Tunesia for well-known customers and accompanies them with regard to the methodical and tool-technical implementation of model-based development processes. One focus is the application of model-based development processes in the area of embedded systems.

The main areas of knowledge are domain-specific modelling methods, behavioural modelling & simulation based on different concepts like state machines, component-based modelling and interface definition languages, code generation, model analysis, artefact traceability for tracking requirements, requirements management, support for industry-specific standards such as AUTOSAR, ReqIF, ISO26262

itemis develops the technical infrastructure for building modelling tools based on various technologies like Eclipse EMF, Xtext, GEF and Xtend, or Jetbrains MPS with extensions like mbeddr. In this role itemis AG provides basic technologies for the implementation of textual and graphical modelling languages. In CrESt, itemis AG focussed on the tool-technical aspects of the research project. In EC1 and EC2, the focus is on the modelling of system architectures. In the MQ3, various cross-cutting aspects of the required tool platform like artefact management and co-simulation were considered.

→ www.itemis.com

Model Engineering Solutions GmbH

Model Engineering Solutions GmbH (MES) is the competence center for model-based software. Divided into the three areas (1) MES Quality Tools, (2) MES Test Center and (3) MES Academy, MES offers its customers optimal support for integrated quality assurance. The MES Quality Tools are the software tools for this. The MES Model Examiner® (MXAM) is the first choice for testing modeling guidelines. The MES Test Manager® efficiently implements requirements-based testing in model-based development. The MES Quality Commander® (MQC) is the quality monitoring tool for evaluating the quality and product capability of software and provides decision-relevant key figures during the development of a product. The MES Test Center includes test services from requirements management to the derivation of test specifications, automated test evaluation and quality monitoring. The MES Academy offers training courses and seminars and supports customers with company-specific consulting and service projects in the introduction and improvement of model-based development processes, such as the fulfillment of standards like ISO 26262. MES customers include well-known OEMs and suppliers to the automotive industry and

www.dbooks.org

customers from the automation technology sector worldwide. MES is a TargetLink® Strategic Partner of dSPACE GmbH and a product partner of MathWorks and ETAS.

→ www.model-engineers.com

OFFIS e.V.

The OFFIS - Institute for Information Technology was founded in 1991 and is an An-Institute of the University of Oldenburg through a cooperation agreement. Its members are the state of Lower Saxony, the University of Oldenburg as well as professors of the Department of Computer Science and computer science related fields. OFFIS is an application-oriented research and development institute, as a "Center of Excellence" for selected topics in computer science and its application areas. OFFIS focuses its research and development activities on IT systems in the application areas of transportation, health, energy and production. The turnover amounts to over 12 million Euro.

The CrESt project involves the R&D area of transportation, which currently comprises about 60 scientific employees. OFFIS has a total of about 260 employees. The Transportation division focuses its research on methods, tools and technologies in the application field of transportation systems for the development of IT-based reliable, cooperative and supporting systems and their ability to interact and collaborate with people intuitively and efficiently. The Transport R&D area comprises several research groups and combines a broad spectrum of competencies in the fields of cognitive psychology, systems and software engineering, electrical engineering and planning theory. Research focuses on methods, processes and tools for establishing the safety of traffic systems as well as methods for the design and analysis of E/E architectures.

OFFIS is or was involved in relevant BMBF projects and European projects within the framework of H2020, Ecsel and ITEA, among others SPES 2020, SPES_XT, ARAMIS I+II, CRYSTAL, DANSE, MBAT, COMBEST, ArtistDesign, AMALTHEA4public, ASSUME, SAFE, PANORAMA, CyberFactory#1, VVMethoden, Set Level 4to5, KI-Delta Learning, and KI-Wissen. OFFIS is a member of ASAM and contributes concepts of Traffic Sequence Charts (TSC) to the standardization of OpenScenario and OpenDrive. Through SafeTRANS OFFIS is also a member of EICOSE, the ARTEMIS Innovation Cluster on Transportation. OFFIS is member (Chamber B) of ARTEMISIA.

Within CrESt, OFFIS participates in the topics "Architectures for Adaptive Systems" and "Open Context". One focus is the deepening of understanding open and dynamic context, and architecture design for adaptive systems. OFFIS contributes to the project with the following competences: (1) model-based design methods for safety-critical embedded systems with supporting analysis techniques especially for the aspects real-time and safety (safety analyses), (2) modelling and analysis of adaptive systems and SoS, (3) validation of human-machine cooperation, and (4) risk analyses as well as

architectural principles and safety architectures under consideration of the aspect safety. OFFIS also participates in the modelling and simulation of human-machine interaction in context-sensitive system networks.

→ www.offis.de

PikeTec GmbH

PikeTec GmbH specializes in the testing and verification of embedded software. With its methods and tools, it significantly simplifies the creation of test cases for embedded systems. Since 2007, PikeTec has therefore been developing and marketing the TPT (Time Partition Testing) test tool. With TPT, tests can be modeled and automatically generated intuitively and flexibly, from simple module tests in MATLAB and Simulink or TargetLink to complex system tests for the vehicle. Tests created with TPT can be reused throughout the development process. TPT is applicable and qualified in the context of the functional safety standards ISO 26262, IEC 61508, EN50128 and DO-178C. The company also accompanies future-oriented software development projects for technical control systems in the form of consulting and engineering services. PikeTec's customers include renowned manufacturers such as VW, Daimler, Bosch and Renault.

→ www.piketec.com

pure-systems GmbH

pure-systems is the leading provider of highly innovative software technologies and solutions for Variant Management and Product Line Engineering (PLE). The company helps their customers increase engineering efficiency through systematic reuse of software engineering assets and reduce product time to market by managing complexity of features & dependencies across systems and variants.

pure::variants, as a Standard Enterprise Solution for PLE, provides deep analytic insights into variants, and can deal with both structural and parametric variability, integrating and supporting diverse authoring tools and engineering assets, like requirements, test cases, architecture & model-based development, source code, documentation, Excel feature lists, among others. As a platform solution, pure::variants provides enterprise scalability and public open APIs, while supporting standards like OSLC, VEL (Variability Exchange Language), Eclipse, EMF, AUTOSAR, and etc.

Today, the variant management solutions from pure-systems are deployed and used successfully with Enterprise Customers in the segments of Automotive, Avionics & Aerospace, Defense & Security, Industry Automation & Production, Rail & Transportation and Semiconductor. The training and consulting services by pure-systems are offered world-wide with the objective of lasting improvement to system

www.dbooks.org

development processes. Typical projects cover issues of requirements, configuration and variant management as well as software architecture and software design.

As product lines and variant management is a relatively new field, continuous research and development is an important part of pure-systems' strategy. Hence, since 2006 pure-systems has also been actively involved in national and European research funding projects (SAFE, ESPA, feasiPLE, DIVa, VARIES, SPES XT, ReVAMP^2, INLIVE, CrESt) and has supported a number of research projects by providing resources (CESAR, AMPLE, ATESST2, MOBILSOFT, VIVASYS, CRYSTAL).

→ www.pure-systems.com

Robert Bosch GmbH

The Bosch Group is a leading global supplier of technology and services. It employs roughly 400,000 associates worldwide (as of December 31, 2019). The company generated sales of 77.7 billion euros in 2019. Its operations are divided into four business sectors: Mobility Solutions, Industrial Technology, Consumer Goods, and Energy and Building Technology. As a leading IoT provider, Bosch offers innovative solutions for smart homes, Industry 4.0, and connected mobility. Bosch is pursuing a vision of mobility that is sustainable, safe, and exciting. It uses its expertise in sensor technology, software, and services, as well as its own IoT cloud, to offer its customers connected, cross domain solutions from a single source. The Bosch Group's strategic objective is to facilitate connected living with products and solutions that either contain artificial intelligence (AI) or have been developed or manufactured with its help. Bosch improves quality of life worldwide with products and services that are innovative and spark enthusiasm. In short, Bosch creates technology that is "Invented for life." The Bosch Group comprises Robert Bosch GmbH and its roughly 440 subsidiary and regional companies in 60 countries. Including sales and service partners, Bosch's global manufacturing, engineering, and sales network covers nearly every country in the world. The basis for the company's future growth is its innovative strength. Bosch employs some 72,600 associates in research and development at 126 locations across the globe, as well as roughly 30,000 software engineers.

→ www.bosch.com

RWTH Aachen University

The RWTH Aachen University (RWTH), established in 1870, is divided into nine faculties. Currently around 45,000 students are enrolled in over 150 academic programs. The number of foreign students (8556) substantiates the university´s international orientation. Every year, more than 6,000 graduates and 800 doctoral

graduates leave the university. 539 professors as well as 5,894 academic and 2,750 non-academic colleagues work at RWTH University.

The research focus of the Chair of Software Engineering at RWTH Aachen University is the definition and improvement of methods for efficient software development. Current fields of research include model-based or generative software development and cyberphysical systems (CPS). The MontiCore language framework developed at the chair allows the agile and compositional development of modeling languages, as well as their use for analysis, synthesis, and generative software development. Based on MontiCore, further languages and tools for the model-driven development of software from the different domains were developed. MontiArc, a modeling language for hierarchical architectures such as CPS, is particularly noteworthy in this context. It also allows the behavior of individual components to be specified via embedded languages (e.g. statecharts). In the field of automotive software engineering, the chair has a long history of research projects and industrial cooperations with large OEMs. The content of these projects covers the whole range of topics from requirements elicitation as well as function, version and variant modeling to software and hardware architecture as well as its use to support analysis and synthesis activities. A prominent use case is the autonomously driving vehicle Caroline, with which Prof. Rumpe successfully participated in the DARPA Urban Challenge.

The Junior Professorship for Mechatronic Systems for Combustion Engines focusses on the interaction of electronical and mechanical powertrain components with innovative control algorithms. Prof. Dr.-Ing. Jakob Andert heads this interdisciplinary and dynamic field of research, which puts a strong emphasis on software-intensive embedded systems that enable cleaner and more efficient vehicle drive systems. Access to the infrastructure of the Center for Mobile Drives enables the efficient use of synergies and direct interaction with researchers working on various topics related to mobile powertrain technology. Research focuses on electrification and hybridization, electric motors and converters for traction drives, in-cycle combustion control and possibilities of connected and autonomous mobility for the powertrain. Hardware-in-the-loop and real-time co-simulations play a key role in the development of testing and validation methods for the future vehicles, including powertrains as well as ADAS/AD systems of interacting and cooperating vehicles.

→ www.rwth-aachen.de

Siemens AG

Siemens AG is a global powerhouse focusing on the areas of electrification, automation and digitalization. One of the world's largest producers of energy-efficient, resource-saving technologies, Siemens is a leading supplier of systems for power generation and transmission as well as medical diagnosis. In infrastructure and industry solutions the company plays a pioneering role. In more than 200 countries/regions the company has

www.dbooks.org

roughly (fiscal year 2019) 385,000 employees of which 39,600 are in digital jobs. For more than 170 years, Siemens stands for technological excellence, innovation, quality, reliability and internationality.

With 2,550 employees worldwide – of which 1,700 are doing research and 300 being engaged in Cybersecurity alone – Corporate Technology (CT) meanwhile since 1905 plays a key role in R&D at Siemens. In research centers located in many different countries, CT works closely with the R&D teams in the Siemens´ Divisions. The CT organization provides expertise regarding strategically important areas to ensure the company's technological future, and to acquire patent rights that safeguard the company's business operations. Against the background of megatrends such as climate change, urbanization, globalization, digitalization and demographic change, CT focuses on innovations that have the potential to change the rules of the game over the long term in business areas that are of interest to Siemens.

CT covers a wide range of technology fields including software and systems innovation, simulation and digital twin, and internet of things, which actively contributed to CrESt.

→ www.siemens.com

Technical University of Kaiserslautern

The work carried out at the chair Software Engineering: Dependability (SEDA) of the Technical University of Kaiserslautern (TUK) is focused mainly on techniques for the development and safety assurance of dependable embedded systems. Current research projects address the improvement and automation of model-based techniques in this field as well as dynamic risk assessment and safety assurance under uncertainty.

The chair was involved in the BMBF-funded projects ARAMiS and ARAMiS II as well as the EU-funded EMC^2 project of the ARTEMIS network. The work is carried out to a large extent in cooperation with partners from the industry. The solutions and tools developed are successfully applied in various domains (e.g. avionics, automotive, commercial vehicles, rail transport). The transfer of the knowledge gained into specialized lectures and theses results in a sustainable strengthening of education.

The SEDA chair was previously involved in the research projects SPES 2020 and SPES_XT. The work performed and results obtained in these projects provided an excellent basis for the work within CrESt. The main contribution is a new concept for the development of dependable collaborative embedded systems that addresses the challenges arising from a highly dynamic and uncertain environment and open context.

→ www.uni-kl.de

Technical University of Munich

The Technical University of Munich (TUM) is one of Europe's top universities. It is committed to excellence in research and teaching, interdisciplinary education and the active promotion of promising young scientists. The university also forges strong links with companies and scientific institutions across the world. TUM was one of the first universities in Germany to be named a University of Excellence. Moreover, TUM regularly ranks among the best European universities in international rankings. In CrESt the chair Software & Systems Engineering (Prof. Broy / Prof. Pretschner) was engaged.

Research and teaching of the Software & Systems Engineering Research Group address central topics of software and systems development. These include basics, methods, processes, models, description techniques and tools. Research focuses on the development of safety-critical embedded systems, mobile and context-adaptive software systems, and development methods for powerful industrially applicable software systems. This is supported by numerous research relevant tools. Research in the field of theorem provers aims at the fundamentals of software engineering. The results and work of our chair have been proven in numerous industrial cooperations. They are successfully applied in telecommunications, avionics, automotive engineering, banking and business information systems. The research group is involved in a wide range of fundamental and application-oriented research projects. In addition, we also provide targeted consulting services to companies, develop prototypes and demonstrators.

→ www.tum.de

Technische Universität Berlin – Daimler Center for Automotive Information Technology Innovations (DCAITI)

The Daimler Center for Automotive Information Technology Innovations (DCAITI) at the Technische Universität Berlin specializes in future scenarios for automotive electronics. The institute was founded in 2006 as a joint initiative of Daimler AG and the Technische Universität Berlin. On the university campus in the historic Telefunken high-rise building on Ernst-Reuter-Platz, various groups of computer specialists and electrical engineers work together in pre-competitive research projects to develop new hardware and software systems for the vehicles of tomorrow. By collaborating with engineering groups from Daimler AG and faculty teams from the Technical University of Berlin as well as selected Fraunhofer Institutes, DCAITI aims to investigate IT-driven product and process innovations for the automotive sector. While some projects design new driver assistance and warning systems, others are concerned with improving the software development process for in-vehicle systems. Several of these projects are part of larger national and pan-European research

www.dbooks.org

initiatives. The DCAITI staff participates in the academic life at the Technische Universität Berlin through teaching assignments and student support. The center encourages students to participate in its projects and gain first-hand experience in automotive electronics in the center's own garage.

→ www.tu-berlin.de

Technische Universität Braunschweig

The TU Braunschweig is one of the TU9 universities and is located with many other research institutes in Europe's strongest research region. In total, the TU Braunschweig has about 18,000 students and about 3500 employees. The Institute for Software Engineering and Automotive Information Technology (ISF) at TU Braunschweig has been headed by Prof. Dr.-Ing. Ina Schaefer since its foundation in July 2012. The goal of the research work at ISF is the development of methods and techniques to increase software quality and to improve efficiency in software development. Application areas for the research results are information systems and embedded systems, especially in the automotive sector. In particular, new concepts and methods are developed in order to design software systems in a way that they can be maintained efficiently and easily extended with new functionalities despite their high complexity. A special research focus is the modeling, implementation and analysis of variant-rich and long-lived software systems. Within the framework of the DFG priority program "Design for Future", ISF is engaged in the development of scalable modeling and analysis concepts for durable, variant-rich automation systems. Within the DFG research group "Controlling Concurrent Change" at the TU Braunschweig, the ISF researches validation methods for evolving embedded systems. In the Electromobility Showcase funded by the BMBS, the ISF is involved in the development of a configurable learning platform for electromobility. Since February 2015, the ISF has contributed to the development of implementation and analysis concepts for evolving variant and context-sensitive embedded systems in the automotive sector within the H2020-ICT project "HyVar". In the CrESt project, the TU Braunschweig will mainly contribute to the topic of modeling and analysis of variability. In addition, TU Braunschweig will apply these concepts to the extraction of flexible system architectures and to the modeling of variability of collaborative systems in a dynamic context.

→ www.tu-braunschweig.de

University of Duisburg-Essen, paluno – The Ruhr Institute for Software Technology

The University of Duisburg-Essen (UDE) is one of the youngest and largest universities in Germany. Since its foundation in 2003, the UDE has developed into a globally recognized research university with a broad spectrum of subjects ranging from humanities, social sciences and education to economics, engineering, natural sciences and medicine. In the latest Times Higher Education Ranking, the UDE holds down 16th place among the 200 best universities worldwide younger than 50 years old. The UDE research institute paluno (The Ruhr Institute for Software Technology) is an association of ten chairs with a total of more than 100 employees. paluno focuses on application-oriented research on software development methods and software technologies for mobile systems, cloud services, big data applications, cyber-physical systems, and self-adaptive systems. The research activities are conducted in close cooperation with partners from industry and research. Key application areas are logistics, mobility, automotive, energy, and production. The researchers of paluno's Software Systems Engineering group (Prof. Pohl) have been and still are significantly involved in numerous research projects. These include, for example, the Big Data Value eCosystem (BDVe), DataPorts (A Data Platform for the Connection of Cognitive Ports), ENACT (Development, Operation, and Quality Assurance of Trustworthy Smart IoT Systems), FogProtect (Protecting Sensitive Data in the Computing Continuum), RestAssured (Secure Data Processing in the Cloud), and TransformingTransport (Big Data Value in Mobility and Logistics) in the Horizon 2020 Programme of the European Union as well as the joint projects SPES_XT, SPES 2020 (Software Platform Embedded Systems 2020), and SPEDIT (Software Platform Embedded Systems Dissemination and Transfer) of the Federal Ministry of Education and Research (BMBF).

→ www.uni-due.de

www.dbooks.org

C – List of Publications

© The Author(s) 2021

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0

A

[Aigner and Grigoleit 2018] C. Aigner, F. Grigoleit: Maintaining configuration knowledge bases: Classification and detection of faults. In: 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), Porto, Portugal, 2018, pp. 33-40.

[Akili 2019] S. Akili: On the Need for Distributed Complex Event Processing with Multiple Sinks. In: 13th ACM International Conference on Distributed and Event-Based Systems (DEBS), Darmstadt, Germany, 2019.

[Akili and Lorenz 2019] S. Akili, F. Lorenz: Towards runtime verification of collaborative embedded systems. In: SICS Software-Intensive Cyber-Physical Systems, 2019, pp. 225-236.

[Akili and Völlinger 2019] S. Akili, K. Völlinger: Case study on certifying distributed algorithms: reducing intrusiveness. In: International Conference on Fundamentals of Software Engineering, Springer, Cham, 2019.

[Al-Hajjaji et al. 2018] M. Al-Hajjaji, M. Schulze, U. Ryssel: Similarity Analysis of Product-Line Variants. In: Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1 (SPLC), ACM, New York, NY, USA, 2018, pp. 226-235.

[Al-Hajjaji et al. 2019] M. Al-Hajjaji, M. Schulze, U. Ryssel: Validating Partial Configurations of Product Lines. In: Proceedings of 13th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), ACM, New York, NY, USA, 2019, pp. 1-6.

[Amorim et al. 2019] T. Amorim, A. Vogelsang, F. Pudlitz, P. Gersing, J. Philipps: Strategies and Best Practices for Model-based Systems Engineering Adoption in Embedded Systems Industry. In: 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada, 2019, pp. 203-212.

[Arai and Schlingloff 2017] R. Arai, H. Schlingloff: Model-based Performance Prediction by Statistical Model Checking: An Industrial Case Study of Autonomous Transport Robots. In: Proceedings of the 25th International Workshop on Concurrency, Specification and Programming, Warsaw, Poland, 2017.

B

[Bandyszak and Brings 2018] T. Bandyszak, J. Brings: Herausforderungen bei der modellbasierten Entwicklung kollaborierender cyber-physischer Systeme für das RE. In: Requirements Engineering Conference (REConf) 2018, HOOD Group, München, 2018.

[Bandyszak et al. 2018] T. Bandyszak, M. Daun, B. Tenbergen, T. Weyer: Model-based Documentation of Context Uncertainty for Cyber-Physical Systems. In: 14th IEEE International Conference on Automation Science and Engineering (CASE), Munich, Germany, 2018.

[Bandyszak et al. 2018] T. Bandyszak, P. Kuhs, J. Kleinblotekamp, M. Daun: On the Use of Orthogonal Context Uncertainty Models in the Engineering of Collaborative Embedded Systems. In: Workshop zur Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018.

[Bandyszak et al. 2020] T. Bandyszak, M. Daun, B. Tenbergen, P. Kuhs, S. Wolf, T. Weyer: Orthogonal Uncertainty Modeling in the Engineering of Cyber-Physical Systems. In: IEEE Transactions on Automation Science and Engineering, Vol. 17, No. 3, 2020, pp. 1250-1265.

[Bandyszak et al. 2020b] T. Bandyszak, T. Weyer, M. Daun: Uncertainty Theories for Real-Time Systems. In: Yuchu Tian, David Charles Levy (eds.): Handbook of Real-Time Computing, Springer, in press, 2022.

[Becker 2020] J.S. Becker: Partial Consistency for Requirement Engineering with Traffic Sequence Charts. In: Software Engineering (Workshops), 2020.

www.dbooks.org

[Bhat et al. 2018] M. Bhat, K. Shumaiev, K. Koch, U. Hohenstein, A. Biesdorf, F. Matthes: An expert recommendation system for design decision making - Who should be involved in making a design decision? In: IEEE International Conference on Software Architecture (ICSA), Seattle, WA, USA, 2018.

[Böhm et al. 2018] B. Böhm, M. Zeller, J. Vollmar, S. Weiß, K. Höfig, V. Malik, S. Unverdorben, C. Hildebrandt: Challenges in the engineering of adaptable and flexible industrial factories. In: Workshop zur Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018.

[Böhm et al. 2020] B. Böhm, J. Vollmar, S. Unverdorben, A. Calà, S. Wolf: Holistic Model-Based Design of System Architectures for Industrial Plants. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2020.

[Böhm et al. 2020b] W. Böhm, D. Mendez Fernandez, et al.: Dealing with Non-Functional Requirements in Model-Driven Development: A Survey. IEEE Transactions on Software Engineering, submitted, 2020.

[Brings 2017] J. Brings: Verifying Cyber-Physical System Behavior in the Context of Cyber-Physical System-Networks. In: 25th IEEE International Requirements Engineering Conference (RE), Lisbon, Portugal, 2017, pp. 556-561.

[Brings and Daun 2019] J. Brings, M. Daun: Towards goal modeling and analysis for networks of collaborative cyber-physical systems. In: ER-Forum at 38th International Conference on Conceptual Modeling (ER), 2019, pp. 70-83.

[Brings and Daun 2020] J. Brings, M. Daun: Towards automated safety analysis for architectures of dynamically forming networks of cyber-physical systems. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW), 2020.

[Brings et al. 2018] J. Brings, M. Daun, C. Hildebrandt, S. Törsleff: An Ontological Context Modeling Framework for Coping with the Dynamic Contexts of Cyber-physical Systems. In: 6th International Conference on Model-Driven Engineering and Software Development, 2018, pp. 396-403.

[Brings et al. 2018b] J. Brings, M. Daun, M. Kempe, T. Weyer: On Different Search Methods for Systematic Literature Reviews and Maps: Experiences from a Literature Search on Validation and Verification of Emergent Behavior. In: 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE), 2018, pp. 35-45.

[Brings et al. 2018c] J. Brings, M. Daun, S. Brinckmann, K. Keller, T. Weyer: Approaches, success factors, and barriers for technology transfer in software engineering – Results of a systematic literature review. In: Software Evolution and Process, vol. 30(11), 2018.

[Brings et al. 2019] J. Brings, M. Daun, M. Kempe, T. Weyer: Validierung und Verifikation von emergentem Verhalten im Software Engineering – Ergebnisse eines Vergleichs unterschiedlicher Suchmethoden. In: Fachtagung Software Engineering, GI, 2019, pp. 135-136.

[Brings et al. 2019b] J. Brings, M. Daun, T. Bandyszak, V. Stricker, T. Weyer, E. Mirzaei, M. Neumann, J.S. Zernickel: Model-based documentation of dynamicity constraints for collaborative cyber-physcial system architectures: Findings from an industrial case study. Systems Architecture, Vol. 97, 2019, pp. 153-167.

[Brings et al. 2020] J. Brings, M. Daun, K. Keller, P. Aluko Obe, T. Weyer: A systematic map on verification and validation of emergent behavior in software engineering research. In: Future Generation Computer Systems, Vol. 112, 2020, pp. 1010-1037.

[Brings et al. 2020b] J. Brings, M. Daun, T. Weyer, K. Pohl: Goal-based configuration analysis for networks of collaborative cyber-physical systems. In: 35th ACM/SIGAPP Symposium on Applied Computing, 2020, pp. 1387-1396.

[Brings et al. 2020c] J. Brings, M. Daun, T. Weyer, P. Pohl: Analyzing Goal Variability in Cyber-Physical System Networks. In: ACM/SIGAPP Applied Computing Reviews, Vol. 21, 2020.

[Bures et al. 2017] T. Bures, D. Weyns, B. Schmerl, J. Fitzgerald, F. Alrimawi, B. Craggs, T. Gabor, I. Gerostathopoulos, D. Liu, F. Murr, B. Nuseibeh, J. Ollesch, J. Ore, L. Pasquale, M. Zasadzinski: Engineering for Smart Cyber-Physical Systems: Report from SEsCPS 2017. In: ACM SIGSOFT Software Engineering Notes, 2017, pp. 19-24.

[Bures et al. 2019] T. Bures, D. Weyns, B.R. Schmerl, J.S. Fitzgerald, A. Aniculaesei, C. Berger, J. Cambeiro, J. Carlson, S.A. Chowdhury, M. Daun, N. Li, M. Markthaler, C. Menghi, B. Penzenstadler, A.D. Pettit, R.G. Pettit IV, L. Sabatucci, C. Tranoris, H. Vangheluwe, S. Voss, E. Zavala: Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2018) - Workshop Report. ACM SIGSOFT Software Engineering Notes 44(4), 2019, pp. 11-13.

[Bures et al. 2020] T. Bures, I. Gerostathopoulos, P. Hnetynka, F. Plasil, F. Krijt, J. Vinarek, J. Kofron: A Language and Framework for Dynamic Component Ensembles in Smart Systems. In: International Journal on Software Tools for Technology Transfer, Springer, 2020, p. 497-509.

[Butting et al. 2017] A. Butting, R. Heim, O. Kautz, J. Ringert, B. Rumpe, A. Wortmann: A Classification of Dynamic Reconfiguration in Component and Connector Architecture Description Languages. In: Proceedings of MODELS 2017 Satellite Event. Workshop ModComp, Austin, Texas, CEUR Workshop Proceedings, 2017.

[Butting et al. 2018] A. Butting, R. Eikermann, O. Kautz, B. Rumpe, A. Wortmann: Controlled and Extensible Variability of Concrete and Abstract Syntax with Independent Language Features. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS), Madrid, Spain, 2018, pp. 75-82.

[Butting et al. 2018b] A. Butting, R. Eikermann, O. Kautz, B. Rumpe, A. Wortmann: Modeling Language Variability with Reusable Language Components. In: Proceedings of the 22nd International Conference on Systems and Software Product Line - Volume 1 (SPLC), Gothenburg, Sweden, 2018, pp. 65-75.

[Butting et al. 2018c] A. Butting, S. Hillemacher, B. Rumpe, D. Schmalzing, A. Wortmann: Shepherding Model Evolution in Model-Driven Development. In: Workshop zur Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018.

[Butting et al. 2018d] A. Butting, S. Konar, B. Rumpe, A. Wortmann: Teaching Model-based Systems Engineering for Industry 4.0: Student Challenges and Expectations. In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (EduSymp@MODELS'18), Copenhagen, Denmark, 2018, pp. 74-81.

[Butting et al. 2019] A. Butting, R. Eikermann, O. Kautz, B. Rumpe, A. Wortmann: Systematic Composition of Independent Language Features. In: Journal of Systems and Software, 152, 2019, pp. 50-69.

C

[Caesar et al. 2018] B. Caesar, W. Klein, C. Hildebrandt, S. Törsleff, A. Fay, J.C. Wehrstedt: New Opportunities using Variability Management in the Manufacturing Domain during Runtime. In: Schäfer, Karagiannis (Hrsg.): Fachtagung Modellierung 2018, Braunschweig, Germany, 2018.

[Caesar et al. 2019] B. Caesar, F. Grigoleit, S. Unverdorben: (Self-)adaptiveness for manufacturing systems: challenges and approaches. In: SICS Software-Intensive Cyber-Physical Systems, Volume 34, Issue 4, Springer, 2019, pp. 191-200.

[Caesar et al. 2019b] B. Caesar, M. Nieke, A. Köcher, C. Hildebrandt, C. Seidl, A. Fay, I. Schaefer: Context-sensitive reconfiguration of collaborative manufacturing systems. In: 9th IFAC Conference on Manufacturing Modelling, Management and Control (MIM) 2019, Berlin, Germany, 2019.

[Cârlan 2017] C. Cârlan: Living Safety Arguments for Open Systems. In: 28th International Symposium on Software Reliability Engineering Workshops (ISSREW), Toulouse, France, 2017, pp. 120-123.

www.dbooks.org

[Cârlan et al. 2019] C. Cârlan, V. Nigam, A. Tsalidis, S. Voss: ExplicitCase: Tool-support for Creating and Maintaining Assurance Arguments Integrated with System Models. In: Proceedings of 9th IEEE International Workshop on Software Certification (WoSoCer), 2019.

[Cioroaica 2019] E. Cioroaica: (Do Not) Trust in Ecosystems. In: 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Montreal, QC, Canada, 2019, pp. 9-12.

[Cioroaica et al. 2018] E. Cioroaica, T. Kuhn, T. Bauer: Prototyping Automotive Smart Ecosystems. In: 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Luxembourg City, 2018, pp. 255-262.

[Cioroaica et. al 2019] E. Cioroaica, S, Chren, B. Buhnova, T. Kuhn, D. Dimitrov: Towards creation of a reference architecture for trust-based digital ecosystems. In: Proceedings of the 13th European Conference on Software Architecture-Volume 2, 2019, pp. 273-276.

[Cioroaica et al. 2019b] E. Cioroaica, F. Pudlitz, I. Gerostathopoulos, T. Kuhn: Simulation Methods and Tools for Collaborative Embedded Systems. In: SICS Software-Intensive Cyber-Physical Systems, Vol. 34, Springer, 2019, pp. 213-223.

[Cioroaica et al. 2020] E. Cioroaica, B. Buhnova, T. Kuhn, D. Schneider: Building Trust in the Untrustable. In: 42nd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), 2020 [In Press.]

[Cioroaica et al. 2020b] E. Cioroaica, S. Chren, B. Buhnova, T. Kuhn, D. Dimitrov: Reference Architecture for Trust-Based Digital Ecosystems. In: International Conference on Software Architecture Companion (ICSA-C), 2020, pp. 266-273.

D

[Dalibor et al. 2019] M. Dalibor, N. Jansen, J. Michael, B. Rumpe, A. Wortmann: Towards Sustainable Systems Engineering-Integrating Tools via Component and Connector Architectures. In: Antriebstechnisches Kolloquium 2019: Tagungsband zur Konferenz, 2019, pp. 121-133.

[Damm et al. 2018] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, A. Rakow: Traffic sequence charts - a visual language for capturing traffic scenarios. In: Embedded Real Time Software and Systems (ERTS), 2018.

[Daun 2018] M. Daun: Using Dedicated Review Models to Support the Validation of Highly Collaborative Systems. Eingeladener Vortrag, Lorentz-Center Workshop zu Dynamics of Multi Agent Systems, Leiden Netherlands, 2018.

[Daun 2019] M. Daun, J. Brings, P. Aluko Obe, S. Weiß, B. Böhm, S. Unverdorben: Using View-Based Architecture Descriptions to Aid in Automated Runtime Planning for a Smart Factory. In: IEEE International Conference on Software Architecture Companion (ICSA-C), Hamburg, Germany, 2019, pp. 202-209.

[Daun and Tenbergen 2020] M. Daun, B. Tenbergen: Teaching Requirements Engineering with Industry Case Examples. In: Tagungsband des 17. Workshops "Software Engineering im Unterricht der Hochschulen", 2020, pp. 49-50.

[Daun et al. 2017] M. Daun, J. Brings, T. Weyer: On the Impact of the Model-Based Representation of Inconsistencies to Manual Reviews - Results from a Controlled Experiment. In: Proceedings of 36th International Conference on Conceptual Modeling (ER), Valencia, Spain, 2017, pp. 466-473.

[Daun et al. 2018] M. Daun, J. Brings, T. Weyer: A Semi-Automated Approach to Foster the Validation of Collaborative Networks of Cyber-Physical Systems. In: 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS), Gothenburg, Sweden, 2018, pp. 6-12.

[Daun et al. 2019] M. Daun, B. Tenbergen, J. Brings, P. Aluko Obe: Sichtenbasierte Kontextmodellierung für die Entwicklung kollaborativer cyber-physischer Systeme. In: Fachtagung Software Engineering, GI, 2019, pp. 123-124.

[Daun et al. 2019b] M. Daun, J. Brings, K. Keller, S. Brinckmann, T. Weyer: Erfolgreicher Technologietransfer im Software Engineering – Transferansätze, Erfolgsfaktoren und Fallstricke. In: Fachtagung Software Engineering, GI, 2019, pp. 135-136.

[Daun et al. 2019c] M. Daun, J. Brings, L. Krajinski, T. Weyer: On the benefits of using dedicated models in validation processes for behavioral specifications. In: International Conference on Software and System Processes (ICSSP), 2019, pp. 44-53.

[Daun et al. 2019d] M. Daun, T. Weyer, K. Pohl: Improving manual reviews in function-centered engineering of embedded systems using a dedicated review model. In: Software and Systems Modeling, vol. 18(6) Springer, 2019, pp. 3421-3459.

[Daun et al. 2019e] M. Daun, V. Stenkova, L. Krajinski, J. Brings, T. Bandyszak, T. Weyer: Goal Modeling for Collaborative Groups of Cyber-Physical Systems with GRL. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 2019, pp. 1600-1609.

[Daun et al. 2020] M. Daun, J. Brings, P. Aluko Obe, K. Pohl, S. Moser, H. Schumacher, M. Rieß.: An Online Course for Teaching Model-based Engineering. In: Tagungsband des 17. Workshops "Software Engineering im Unterricht der Hochschulen", 2020, pp. 66-67.

[Daun et al. 2020b] M. Daun, J. Brings, T. Weyer: Do Instance-level Review Diagrams Support Validation Processes of Cyber-Physical System Specifications Results from a Controlled Experiment. In: International Conference on Software and Systems Process (ICSSP), Seoul, Republic of Korea. ACM, New York, NY, USA, 2020.

[Daun et al. 2020c] M. Daun, T. Weyer, K. Pohl: Ein Review-Modell zur Unterstützung in der funktionszentrierten Entwicklung eingebetteter Systeme. In: Proceedings of the Tagung Software Engineering, GI, 2020, p. 39-40.

G

[Gerostathopoulos et al. 2018] I. Gerostathopoulos, C. Prehofer, T. Bures: Adapting a System with Noisy Outputs with Statistical Guarantees. In: 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2018, pp. 58-68.

[Gerostathopoulos et al. 2018b] I. Gerostathopoulos, C. Prehofer, L. Bulej, T. Bures, V. Horky, P. Tuma: Cost-Aware Stage-Based Experimentation: Challenges and Emerging Results. In: EEE International Conference on Software Architecture Companion (ICSA-C), Seattle, WA, USA, 2018, pp. 72-75.

[Gerostathopoulos et al. 2018c] I. Gerostathopoulos, C. Prehofer, J. Thomas, B. Bischl: Online Experiment-Driven Adaptation. Submitted to IEEE Software, 2018.

[Gerostathopoulos et al. 2018d] I. Gerostathopoulos, C. Prehofer, A. Uysal, T. Bures: A Tool for Online Experiment-Driven Adaptation. In: 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, Italy, 2018, pp. 100-105.

[Gerostathopoulos et al. 2019] I. Gerostathopoulos, M. Konersmann, S. Krusche, D. I. Mattos: Continuous Data-driven Software Engineering – Towards a Research Agenda. SIGSOFT Software Engineering Notes 44, 3, 2019, pp. 60–64.

[Gerostathopoulos et al. 2019b] I. Gerostathopoulos, S. Kugele, C. Segler. T. Bures, A. Knoll: Automated Learnability Evaluation for Smart Automotive Software Functions. In: 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019.

www.dbooks.org

[Greifenberg 2019] T. Greifenberg: Artefaktbasierte Analyse modellgetriebener Softwareentwicklungsprojekte. In: Aachener Informatik-Berichte, Software Engineering, Band 42, Shaker Verlag, 2019.

H

[Habtom et al. 2019] K. Habtom, A. Collins, D. Marmsoler: Modeling and Verifying Dynamic Architectures with FACTum Studio. In: International Conference on Formal Aspects of Component Software, 2019, pp. 243-251.

[Hayward et al. 2020] A. Hayward, M. Daun, W. Böhm, A. Petrvoska, L. Krajinski, A. Fay: Modellierung von Funktionen in der modellbasierten Entwicklung von Systemverbünden kollaborierender cyber-physischer Systeme. In: Entwurf komplexer Automatisierungssysteme (EKA), 2020.

[Hildebrandt et al. 2018] C. Hildebrandt, S. Törsleff, B. Caesar, A. Fay: Ontology Building for cyber-physical-systems: From Requirements to heavyweight Ontologies. In: 14th IEEE International Conference on Automation Science and Engineering (CASE), Munich, Germany, 2018.

[Hildebrandt et al. 2018b] C. Hildebrandt, S. Törsleff, T. Bandyszak, B. Caesar, A. Ludewig, A. Fay: Ontology Engineering for Collaborative Embedded Systems – Requirements and Initial Approach. In: Workshop zur Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018.

[Hildebrandt et al. 2018c] C. Hildebrandt, W. Klein, J.C. Wehrstedt, A. Fay: Ontology-based Simulation of Manufacturing Systems in Open and Dynamic Contexts. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2018.

[Hildebrandt et al. 2019] C. Hildebrandt, T. Bandyszak, A. Petrovska, N. Laxman, E. Cioroaica, S. Törsleff: EURECA: epistemic uncertainty classification scheme for runtime information exchange in collaborative system groups, In: SICS Software-Intensive Cyber-Physical Systems, Vol. 34, Springer, 2019, pp. 177–190.

[Hinterreiter et al. 2019] D. Hinterreiter, M. Nieke, L. Linsbauer, C. Seidl, H. Prähofer, P. Grünbacher: Harmonized temporal feature modeling to uniformly perform, track, analyze, and replay software product line evolution. In: Proceedings of the 18th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE). ACM, New York, NY, USA, 2019, pp. 115–128.

[Hipp et al. 2020] U. Hipp, T. Zeh, W. Klein, A. Joanni, S. Rothbauer, M. Zeller: Simulation-based robust scheduling for smart factories considering improved test strategies. RAMS 2020, 2020.

[Hnetynka et al. 2018] P. Hnetynka, P. Kubat, R. Al-Ali, I. Gerostathopoulos, D. Khalyeyev: Guaranteed Latency Applications in Edge-Cloud Environment. In: 2nd Context-aware, Autonomous and Smart Architectures International Workshop (CASA), 2018.

[Hoang et al. 2019] X.-L. Hoang, B. Caesar, A. Fay: Adaptation of Manufacturing Machines by the Use of Multiple-Domain-Matrices and Variability Models. In: 9th IFAC Conference on Manufacturing Modelling, Management and Control (MIM) 2019, Berlin, Germany, 2019.

[Höfig et al. 2019] K. Höfig, C. Klein, S. Rothbauer, M. Zeller, M. Vorderer, C. H. Koo: A Meta-model for Process Failure Mode and Effects Analysis (PFMEA). In: 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019, pp. 1199-1202.

J

[Jöckel and Kläs 2019] L. Jöckel, M. Kläs: Increasing Trust in Data-Driven Model Validation – A Framework for Probabilistic Augmentation of Images and Meta-Data Generation using Application Scope

Characteristics. In: 38th International Conference on Computer Safety, Reliability and Security, SafeComp 2019, Turku, Finland, 2019, pp. 155-164.

[Jöckel et al. 2019] L. Jöckel, M. Kläs, S. Martínez-Fernández: Safe Traffic Sign Recognition through Data Augmentation for Autonomous Vehicles Software. In: 19th IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, 2019, pp. 540-541.

K

[Kaiser et al. 2018] B. Kaiser, D. Schneider, R. Adler, D. Domis, F. Möhrle, A. Berres, M. Zeller, K. Höfig, M. Rothfelder: Advances in Component Fault Trees, Safety and Reliability – Safe Societies in a Changing World. In: Proceedings of 28th European Safety and Reliability Conference (ESREL), Trondheim, Norway, Taylor & Francis (CRC Press), 2018, pp. 815-823.

[Keller et al. 2018] K. Keller, A. Neubauer, J. Brings, M. Daun: Tool-Support to Foster Model-based Requirements Engineering for Cyber-Phsyical Systems. In: Workshop zur Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018, pp. 47-56.

[Keller et al. 2018b] K. Keller, J. Brings, M. Daun, T. Weyer: A Comparative Analysis of ITU-MSC-Based Requirements Specification Approaches Used in the Automotive Industry. In: Proceedings of 10th International Conference on System Analysis and Modeling (SAM), Copenhagen, Denmark, 2018, pp. 183-201.

[Kläs 2018] M. Kläs: Towards Identifying and Managing Sources of Uncertainty in AI and Machine Learning Models-An Overview. arXiv preprint arXiv:1811.11669, 2018.

[Kläs and Sembach 2019] M. Kläs, L. Sembach: Uncertainty Wrappers for Data-driven Models – Increase the Transparency of AI/ML-based Models through Enrichment with Dependable Situation-aware Uncertainty Estimates, 2nd Int. Workshop on Artificial Intelligence Safety Engineering (WAISE 2019), Turku, Finland, 2019.

[Kläs and Vollmer 2018] M. Kläs, A.M. Vollmer: Uncertainty in Machine Learning Applications – A Practice-Driven Classification of Uncertainty. In: First International Workshop on Artificial Intelligence Safety Engineering (WAISE 2018), Västerås, Sweden, 2018.

[Koo et al. 2018] C. H. Koo, M. Vorderer, S. Junker, S. Schröck, A. Verl: Challenges and requirements for the safety compliant operation of reconfigurable manufacturing systems. In: Proceedings CIRP Conference on Manufacturing System, Vol. 72, 2018, pp. 1100-1105.

[Koo et al. 2019] C. H. Koo, M. Vorderer, S. Schröck, J. Richter, A. Verl: Assistierte Risikobeurteilung für wandlungsfähige Plug and Produce Montagesysteme. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2019.

[Koo et al. 2019b] C. H. Koo, S. Rothbauer, M. Vorderer, K. Höfig, M. Zeller: SQUADfps: Integrated model-based machine safety and product quality for flexible production systems. In: 6th International Symposium on Model-Based Safety and Assessment (IMBSA), Thessaloniki, Greece, 2019, pp. 222-236.

[Koo et al. 2020] C. H. Koo, N. Laxman, F. Möhrle: Runtime safety analysis for reconfigurable production systems. In: 30th European Safety and Reliability Conference (ESREL), Venice, Italy, 2020.

[Koo et al. 2020b] C. H. Koo, S. Schröck, M. Vorderer, J. Richter, A. Verl: Assistierte Risikobeurteilung für wandlungsfähige Montagesysteme. In: ATP-Edition, Fachmagazin für Automatisierungstechnische Praxis 05/2020, 2020, pp. 68-75.

[Koo et al. 2020c] C. H. Koo, S. Schröck, M. Vorderer, J. Richter, A. Verl: A model-based and software-assisted safety assessment concept for reconfigurable PnP-systems. In: 53rd CIRP Conference on Manufacturing System, Chicago, USA, 2020.

www.dbooks.org

[Kurpiewski and Marmsoler 2019] D. Kurpiewski, D. Marmsoler: Strategic Logics for Collaborative Embedded Systems: Specification and Verification of Collaborative Embedded Systems using Strategic Logics. In: SICS Software-Intensive Cyber-Physical Systems, Vol. 34, Springer, 2019, pp. 201-212.

L

[Lackner and Schlingloff 2017] H. Lackner, H. Schlingloff: Advances in Testing Software Product Lines. In: Advances in Computers, Vol. 107, Elsevier, 2017, pp. 157-217.

[Laxman et al. 2020] N. Laxman, C. H. Koo, P. Liggesmeyer: U-Map: A reference map for safe handling of runtime uncertainties. In: 7th International Symposium on Model-Based Safety and Assessment (IMBSA), Lisbon, Portugal, accepted, 2020.

[Lorenz and Schlingloff 2018] F. Lorenz, H. Schlingloff: Online-Monitoring Autonomous Transport Robots with an R-valued Temporal Logic. In: 14th IEEE International Conference on Automation Science and Engineering (CASE), Special Session on Engineering Methods and Tools for the Development of Collaboration-intensive Cyber Physical Systems, Munich, Germany, 2018, pp. 1093-1098.

[Ludewig et al. 2018] A. Ludewig, M. Daun, A. Petrovska, W. Böhm, A. Fay: Requirements for Modeling Dynamic Function Networks for Collaborative Embedded Systems. In: Workshop zur Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018.

M

[Marmsoler 2017] D. Marmsoler: Dynamic Architectures. Archive of Formal Proofs, 2017.

[Marmsoler 2017b] D. Marmsoler: On the Semantics of Temporal Specifications of Component-Behavior for Dynamic Architectures. In: 11th International Symposium on Theoretical Aspects of Software Engineering, Sophia Antipolis, 2017, pp. 1-6.

[Marmsoler 2018] D. Marmsoler: A Framework for Interactive Verification of Architectural Design Patterns in Isabelle/HOL. In: Proceedings of 20th International Conference on Formal Engineering Methods (ICFEM), Gold Coast, QLD, Australia, 2018, pp. 12-16.

[Marmsoler 2018b] D. Marmsoler: Hierarchical Specification and Verification of Architecture Design Patterns. In: 21st International Conference on Fundamental Approaches to Software Engineering (FASE), 2018, pp. 149-168.

[Marmsoler 2019] D. Marmsoler: “A Denotational Semantics for Dynamic Architectures”. In: Theoretical Aspects of Software Engineering. 2019.

[Marmsoler 2019b] D. Marmsoler: A Calculus for Dynamic Architectures. In: Science of Computer Programming, 2019.

[Marmsoler 2019c] D. Marmsoler: Axiomatic Specification and Verification of Architectural Design Patterns using Interactive Theorem Proving. Dissertation. 2019.

[Marmsoler 2019d] D. Marmsoler: Composition in Dynamic Architectures based on Fixed Points in Lattices. In: International Colloquium on Theoretical Aspects of Computing, 2019.

[Marmsoler 2019e] D. Marmsoler: Verifying Dynamic Architectures using Model Checking and Interactive Theorem Proving. In: Proceedings Software Engineering und Software Management, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, Germany, 2019.

[Marmsoler and Blakqori 2019] D. Marmsoler, G. Blakqori: APML: An Architecture Proof Modeling Language. In: 23rd International Symposium on Formal Methods, 2019.

[Marmsoler and Degenhardt 2017] D. Marmsoler, S. Degenhardt: Patterns of Dynamic Architectures using Model Checking. In: Proceedings International Workshop on Formal Engineering approaches to Software Components and Architectures, 2017.

[Marmsoler and Gidey 2018] D. Marmsoler, H. K. Gidey: FACTUM Studio: A Tool for the Axiomatic Specification and Verification of Architectural Design Patterns. In: 15th International Conference on Formal Aspects of Component Software (FACS), Pohang, South Korea, 2018, pp. 279-287.

[Marmsoler and Gidey 2019] D. Marmsoler, H.K. Gidey: Interactive Verification of Architectural Design Patterns in FACTum. In: Formal Aspects of Computing. 2019.

[Marmsoler and Habtom 2019] D. Marmsoler, H.K. Gidey: Interactive Verification of Architectural Design Patterns in FACTum. In: Formal Aspects of Computing, 2019.

[Marmsoler and Petrovska 2019] D. Marmsoler, A. Petrovska: Detecting Architectural Erosion using Runtime Verification. In: 12th Interaction and Concurrency Experience (ICE), 2019.

[Marmsoler and Petrovska 2020] D. Marmsoler, A. Petrovska: Detecting Architectural Erosion using Runtime Verification. In: Journal of Logical and Algebraic Methods in Programming (JLAMP), submitted, 2020.

[Marmsoler et al. 2017] D. Marmsoler, D.V. Hung, D. Kapur (Eds.): Towards a Calculus for Dynamic Architectures. In: 14th International Colloquium on Theoretical Aspects of Computing (ICTAC), Hanoi, Vietnam, 2017, pp. 79-99.

[Mauro et al. 2017] J. Mauro, M. Nieke, C. Seidl, I. Chieh Yu: Anomaly Detection and Explanation in Context-Aware Software Product Lines. In: Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC), New York, USA, 2018, pp. 18-21.

[Mauro et al. 2018] J. Mauro, M. Nieke, C. Seidl, I. Chieh Yu: Context-Aware Reconfiguration in Evolving Software Product Lines. In: Science of Computer Programming. Volume 163, 2018.

[Mendez Fernandez et al. 2019] D. Mendez Fernandez, W. Böhm, A. Vogelsang, J. Mund, M. Broy, M. Kuhrmann, T. Weyer: Artefacts in software engineering: a fundamental positioning. In: Software & Systems Modeling, 2019.

[Meyer 2019] M. Meyer: 3D Multi-Vehicle Co-Simulation Framework for Testing of Cooperative Automated Driving Functions. In: FEV Simulation and Calibration Symposium 2019, Stuttgart, 2019.

[Meyer et al. 2020] M. Meyer, C. Granrath, J. Andert, G. Feyerl, J. Richenhagen, J. Kaths: Closed-loop Platoon Simulation with Cooperative Intelligent Transportation Systems based on Vehicle-to-X Communication. Simulation Modelling Practice and Theory, Elsevier, accepted, 2020.

[Meyer et al. 2020b] M. Meyer, C. Granrath, L. Wachtmeister, N. Jäckel: Methoden für die Entwicklung kollaborativer eingebetteter Systeme in automatisierten Fahrzeugen. In: ATZelektronik, vol. 12, Springer, accepted, 2020.

[Ming and Schlingloff 2017] C. Ming, H. Schlingloff: Monitoring with Parametrized Extended Life Sequence Charts. In: Fundamenta Informaticae, Vol. 153(3), IOS Press, 2017, pp. 173-198.

[Möhrle et al. 2017] F. Möhrle, M. Zeller, K. Höfig, M. Rothfelder, P. Liggesmeyer: Towards Automated Design Space Exploration for Safety-Critical Systems Using Type-Annotated Component Fault Trees. In: 5th International Symposium on Model-Based Safety and Assessment (IMBSA), Trento, Italy, 2017.

N

[Nieke et al. 2018] M. Nieke, C. Seidl, T. Thüm: Back to the Future: Avoiding Paradoxes in Feature-Model Evolution. In: Proceedings of the 22nd International Systems and Software Product Line Conference - Volume B (SPLC), ACM, New York, NY, USA, 2018, pp. 48-51.

www.dbooks.org

[Nieke et al. 2018b] M. Nieke, J. Mauro, C. Seidl, T. Thüm, I. Chieh Yu: Anomaly Analyses for Feature-Model Evolution. In: Proceedings of the 17th International Conference on Generative Programming: Concepts and Experiences (GPCE), 2018, pp. 188-201.

[Nieke et al. 2019] M. Nieke, A. Hoff, C. Seidl: Automated metamodel augmentation for seamless model evolution tracking and planning. In Proceedings of the 18th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE). ACM, New York, NY, USA, 2019, pp. 68–80.

P

[Petrovska 2019] A. Petrovska: Semi-distributed architecture for smart self-adaptive cyber-physical systems. In: 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, part of 11st International Conference on Software Engineering (ICSE), 2019.

[Petrovska and Grigoleit 2018] A. Petrovska, F. Grigoleit: Towards Context Modeling for Dynamic Collaborative Embedded Systems in Open Context. In: 10th International Workshop on Modelling and Reasoning in Context (MRC) at International Joint Conference of Artificial Intelligence, Stockholm, Schweden, 2018.

[Petrovska and Pretschner 2019] A. Petrovska, A. Pretschner: Learning Approach for Smart Self-Adaptive Systems. In: 13th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Umea, Sweden, 2019, pp. 234-236.

[Petrovska et al. 2019] A. Petrovska, S. Quijano, I. Gerostathopoulos, A. Pretschner: Knowledge Aggregation with Subjective Logic in Multi-Agent Self-Adaptive Cyber-Physical Systems. In: 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2019, pp. 39-50.

[Pudlitz et al. 2019] F. Pudlitz, A. Vogelsang, F. A. Brokhausen: A Lightweight Multilevel Markup Language for Connecting Software Requirements and Simulations. In: Knauss E., Goedicke M. (eds) Requirements Engineering: Foundation for Software Quality. REFSQ 2019. Lecture Notes in Computer Science, Vol. 11412, Springer, Cham, 2019.

R

[Reich 2018] V. Reich: Development and Evaluation of Decision Strategies for Manufacturing in Industrie 4.0 using Plant Simulation, Masterarbeit, TU München, 2018.

[Rösel 2019] S. Rösel: Guidelines are a Modeler's best friends – ein Einstieg in die statische Modellanalyse. In: Automation Software Engineering Kongress, Sindelfingen, 2019.

[Rösel 2019b] S. Rösel: ISO 26262 in 10 Schritten sicherheitsrelevante Embedded Software erstellen. In: Embedded Software Engineering Kongress, Sindelfingen 2019.

[Rosen 2019] R. Rosen: Digital Twin & Symbiotic Mechatronics Approaches for System Development. Models 2019 (invited speaker).

[Rosiak et al. 2019] K. Rosiak, O. Urbaniak, A. Schlie, C. Seidl, I. Schaefer: Analyzing Variability in 25 Years of Industrial Legacy Software: An Experience Report. In: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B (SPLC), ACM, New York, NY, USA, 2019, pp. 65-72.

[Rumpe et al. 2019] B. Rumpe, I. Schaefer, H. Schlingloff, A. Vogelsang: Special issue on engineering collaborative embedded systems, In: SICS Software-Intensive Cyber-Physical Systems, Vol. 34, Springer, 2019, pp. 173–175.

S

[Schenk et al. 2019] T. Schenk, A.Botero Halblaub, J. C. Wehrstedt: Co-Simulation scenarios in industrial production plants. Industrial User Presentations. In: 13th International Modelica Conference, Regensburg, 2019.

[Schlie et al. 2017] A. Schlie, D. Wille, L. Cleophas, I. Schaefer: Clustering Variation Points in MATLAB/Simulink Models Using Reverse Signal Propagation Analysis. Proceedings of the International Conference on Software Reuse (ICSR), Springer, Salvador, Brazil, 2017.

[Schlie et al. 2018] A. Schlie, S. Schulze, I. Schaefer: Comparing Multiple MATLAB/Simulink Models Using Static Connectivity Matrix Analysis. In: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 2018, pp. 160-171.

[Schlie et al. 2019] A. Schlie, C. Seidl, I. Schaefer: Reengineering Variants of MATLAB/Simulink Software Systems. In: Security and Quality in Cyber-Physical Systems Engineering. Springer International Publishing, 2019, pp. 267-301.

[Schlie et al. 2019b] A. Schlie, K. Rosiak, O. Urbaniak, I. Schaefer, B. Vogel-Heuser: Analyzing Variability in Automation Software with the Variability Analysis Toolkit. In: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B (SPLC), ACM, New York, NY, USA, 2019, pp. 191-198.

[Schlingloff 2018] H. Schlingloff: Specification and Verification of Collaborative Transport Robots. In: 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), Porto, Portugal, 2018, pp. 3-8.

[Schlingloff 2019] H. Schlingloff: PhD, the University, and Everything. In: 30th International Symposium on Software Reliability Engineering (ISSRE), Berlin, Germany, 2019.

[Schlingloff 2019b] H. Schlingloff: Strategy Synthesis. Invited paper at CS&P 2019: 28th International Workshop on Concurrency, Specification, and Programming, Olstyn, Poland, 2019.

[Schlingloff 2019c] H. Schlingloff: Teaching Model Checking via Games and Puzzles. In: Proceedings of 1st International Workshop "Formal Methods - Fun for Everybody" (FMFun), Co-located with iFM 2019, Bergen, Norway, 2019.

[Schmidt 2019] K. Schmidt: Modellierung und Test: Software für Industrie-Transportroboter. Embedded Testing, Munich, Germany, 2019.

[Schulze 2019] C. Schulze: Agile Software-Produktlinienentwicklung im Kontext heterogener Projektlandschaften. In: Aachener Informatik-Berichte, Software Engineering, Band 40, Shaker Verlag, 2019.

[Schuster et al. 2017] S. Schuster, C. Seidl, I. Schaefer: Towards a development process for maturing Delta-oriented software product lines. In Proceedings of the 8th ACM SIGPLAN International Workshop on Feature-Oriented Software Development (FOSD), 2017, p. 41-50.

[Seitz et al. 2018] A. Seitz, D. Henze, D. Miehle, B. Bruegge, J. Nickles, M. Sauer: Fog Computing as Enabler for Blockchain-Based IIoT App Marketplaces – A Case Study. In: 5th International Conference on Internet of Things: Systems, Management and Security (IoTSMS), Valencia, Spain, 2018, pp. 182-188.

[Seitz et al. 2018b] A. Seitz, D. Henze, J. Nickles, M. Sauer, B. Bruegge: Augmenting the Industrial Internet of Things with Emojis. In: Third International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, 2018, pp. 240-245.

[Smirnov et al. 2018] D. Smirnov, T. Schenk, J. C. Wehrstedt, Hierarchical Simulation of Production Systems, In: 14th IEEE International Conference on Automation Science and Engineering (CASE), Munich, Germany, 2018.

www.dbooks.org

[Stenkova et al. 2019] V. Stenkova, J. Brings, M. Daun, T. Weyer: Generic negative scenarios for the specification of collaborative cyber-physical systems. In: Proceedings of 38th International Conference on Conceptual Modeling (ER), 2019, pp. 412-419.

[Stenkova et al. 2020] V. Stenkova, M. Daun, J. Brings, T. Weyer: Generische Negativszenarien in der Entwicklung kollaborativer cyber-physischer Systeme. In: Fachgruppentreffen "Requirements Engineering" der Gesellschaft für Informatik, GI, accepted, 2020.

T

[Tenbergen et al. 2018] B. Tenbergen, M. Daun, P. Aluko Obe, J. Brings: View-Centric Context Modeling to Foster the Engineering of Cyber-Physical System Networks. In: IEEE International Conference on Software Architecture (ICSA), Seattle, WA, USA, 2018.

[Törsleff et al. 2018] S. Törsleff, C. Hildebrandt, M. Daun, J. Brings, A. Fay: Modeling the Dynamic and Open Context of Collaborative Embedded Systems: Requirements and Initial Approach. In: Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), 2018, pp. 25-32.

[Törsleff et al. 2018b] S. Törsleff, C. Hildebrandt, M. Daun, J. Brings, A. Fay: Developing Ontologies for the Collaboration of Cyber-Physical Systems: Requirements and Solution Approach. In: 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), Porto, Portugal, 2018.

[Törsleff et al. 2019] S. Törsleff, C. Hildebrandt, A. Fay: Development of Ontologies for Reasoning and Communication in Multi-Agent Systems. In: 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, 2019.

U

[Unverdorben et al. 2018] S. Unverdorben, B. Böhm, A. Lüder: Reference Architectures for Future Production Systems in the Field of discrete manufacturing. In: 14th IEEE International Conference on Automation Science and Engineering (CASE), Munich, Germany, 2018, pp. 869-874.

[Unverdorben et al. 2019] S. Unverdorben, B. Böhm, A. Lüder: Concept for Deriving System Architectures from Reference Architectures. In: 2019 IEEE International Conference on Industrial Engineering & Engineering Management (IEEM), Macau, 2019.

[Unverdorben et al. 2019b] S. Unverdorben, B. Böhm, A. Lüder: Industrie 4.0 – Architekturansätze und zugehörige Konzepte für konventionelle Produktionsanlagen / Industrie 4.0 – Architectural approaches and related concepts for conventional production systems. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2019.

V

[Velasco Moncada et al.] S. Velasco Moncada, J. Reich, M. Tchangou: Interactive information zoom on Component Fault Trees. In: Schaefer, I., Karagiannis, D., Vogelsang, A., Méndez, D. & Seidl, C. (Hrsg.), Modellierung 2018. Gesellschaft für Informatik e.V., Bonn, 2018, pp. 311-314.

[Velasco Moncada 2020] D. S. Velasco Moncada: Hazard-driven realization views for Component Fault Trees. In: Software and Systems Modeling, Springer, 2020.

W

[Wager and Prehofer 2018] A. Wager, C. Prehofer: Translating Multi-Device Task Models to State Machines. In: 6th International Conference on Model-Driven Engineering and Software Development, SciTePress 2018, pp. 201-208.

[Wehrstedt et al. 2019] J. C. Wehrstedt, B. Groos, W. Klein, V. Malik, S. Rothbauer, M. Zeller, S. Weiß, B. Böhm, J. Brings, M. Daun, B. Caesar, A. Fay, C. H. Koo, M. Vorderer: A Seamless Description Approach for Engineering – Methods Illustrated for Industrie 4.0 Scenarios. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2019.

[Wehrstedt et al. 2020] J. C. Wehrstedt, A. Sohr, T. Schenk, R. Rosen, Y. Zhou: A Framework for Operator Assist Apps of Automated Systems. In: IFAC World Congress, Berlin, 2020.

[Weiß et al. 2018] S. Weiß, B. Böhm, S. Unverdorben, J. Vollmar: Auswirkungen zukünftiger Zusammenarbeitsszenarien auf industrielle Produktionsanlagen. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2018.

[Weiß et al. 2019] S. Weiß, B. Caesar, B. Böhm, J. Vollmar, A. Fay: Modellierung von Fähigkeiten industrieller Anlagen für die auftragsgesteuerte Produktion. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2019.

[Weyer 2018] T. Weyer: Requirements Engineering im Zeitalter von Digitalisierung und Autonomen Systemen. In: Requirements Engineering Conference (REConf) 2018, HOOD Group, München, 2018

[Wolf et al. 2020] S. Wolf, B. Caesar, A. Fay, B. Böhm: Erstellung eines Domänenmodells zur Beschreibung von Fähigkeiten fertigungstechnischer Anlagen für die auftragsgesteuerte Produktion. In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2020.

Z

[Zarras et al. 2018] A. Zarras, I. Gerostathopoulos, D. Mendez Fernandez: Can Today's Machine Learning Pass Image-based Turing Tests? In: 40th International Conference on Software Engineering (ICSE), 2018, pp. 129-148.

[Zarras et al. 2018b] A. Zarras, I. Gerostathopoulos, D. Mendez Fernandez: Shooting Ourselves on the Foot: Can Today's Machine Learning Pass Image-Based Turing Tests? ACM Internet Measurement Conference 2018 (IMC 2018), submitted, 2018.

[Zernickel and Schmiljun 2018] J. S. Zernickel, A. Schmiljun: Die Fabrik der Zukunft, Fachartikel in „Deutsche Verkehrszeitung", 2018.

[Zernickel and Stubert 2017] J. Zernickel, H. Stubert: Podiumsdiskussion zum Thema „Industrie 4.0" mit Vertretern aus Wirtschaft, Wissenschaft und Politik, Berlin, 2017.

[Zhou et al. 2019] Y. Zhou, M. Allmaras, A. Massalimova, T. Schenk, A. Sohr, J.C. Wehrstedt: Assist System Framework for Production Prioritization - Flexible Architecture to integrate Simulation in Run-Time Environment, In: VDI-Kongress AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden, Germany, 2019.

www.dbooks.org