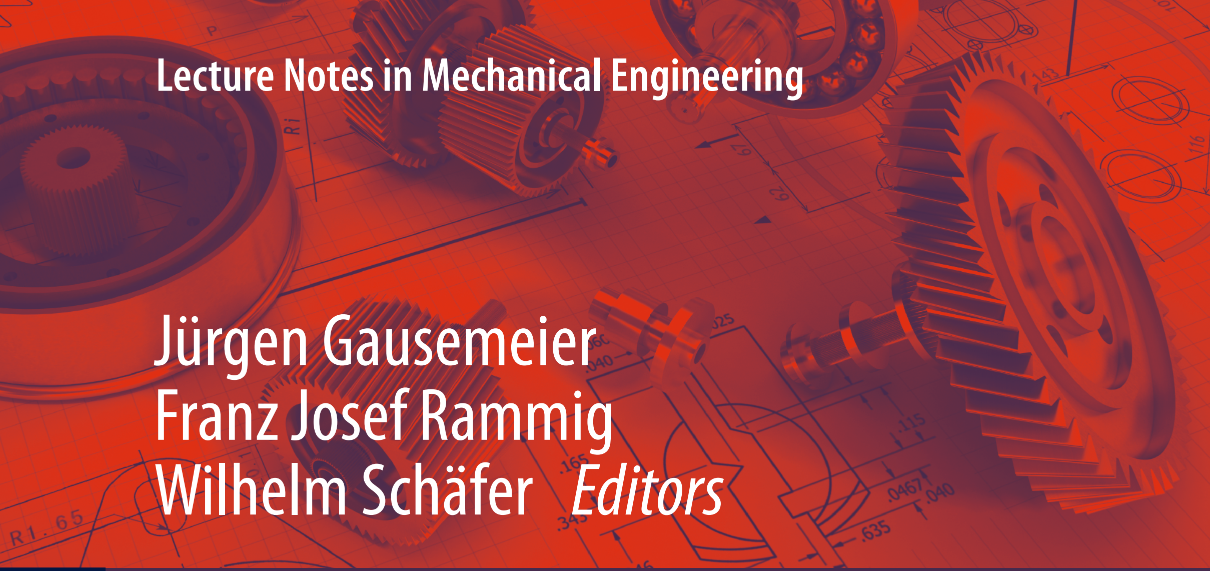
Lecture Notes in Mechanical Engineering



Jürgen Gausemeier
Franz Josef Rammig
Wilhelm Schäfer *Editors*

# Design Methodology for Intelligent Technical Systems

## Develop Intelligent Technical Systems of the Future

Springer

# Lecture Notes in Mechanical Engineering

For further volumes:
http://www.springer.com/series/11236

*About this Series*

Lecture Notes in Mechanical Engineering (LNME) publishes the latest developments in Mechanical Engineering - quickly, informally and with high quality. Original research reported in proceedings and post-proceedings represents the core of LNME. Also considered for publication are monographs, contributed volumes and lecture notes of exceptionally high quality and interest. Volumes published in LNME embrace all aspects, subfields and new challenges of mechanical engineering. Topics in the series include:

- Engineering Design
- Machinery and Machine Elements
- Mechanical Structures and Stress Analysis
- Automotive Engineering
- Engine Technology
- Aerospace Technology and Astronautics
- Nanotechnology and Microengineering
- Control, Robotics, Mechatronics
- MEMS
- Theoretical and Applied Mechanics
- Dynamical Systems, Control
- Fluid Mechanics
- Engineering Thermodynamics, Heat and Mass Transfer
- Manufacturing
- Precision Engineering, Instrumentation, Measurement
- Materials Engineering
- Tribology and Surface Technology

Jürgen Gausemeier · Franz Josef Rammig
Wilhelm Schäfer
Editors

# Design Methodology for Intelligent Technical Systems

## Develop Intelligent Technical Systems of the Future

Springer

*Editors*

Jürgen Gausemeier
Product Engineering
Heinz Nixdorf Institute
University of Paderborn
Paderborn
Germany

Wilhelm Schäfer
Software Engineering
Heinz Nixdorf Institute
University of Paderborn
Paderborn
Germany

Franz Josef Rammig
Design of Distributed Embedded Systems
Heinz Nixdorf Institute
University of Paderborn
Paderborn
Germany

ISSN 2195-4356 ISSN 2195-4364 (electronic)
ISBN 978-3-642-45434-9 ISBN 978-3-642-45435-6 (eBook)
DOI 10.1007/978-3-642-45435-6
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013956527

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.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

# Preface

The Collaborative Research Centre 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering", funded from 2002 to 2013 by the German Research Foundation (DFG) has focused vanguard products for future markets that are characterized by the close interaction of mechatronics with information and communication technology. The CRC 614 combines the greatest strengths of the University of Paderborn and the Heinz Nixdorf Institute, the symbiosis of computer science, engineering and mathematics. This is visible by the strong expertise in the field of mechatronics in conjunction with optimization. Our focus was on mechatronic systems with inherent partial intelligence that have cognitive functions.

Our vision has been a design methodology for such systems. The CRC 614 has been pursuing the long-term aim of opening up the active paradigm of self-optimization for mechanical engineering and also enabling others to develop self-optimizing systems. Facing a large span of aspects and issues, an interdisciplinary working group was funded – the interest group self-optimization (IG SO) – to bring together the many thoughts and basic concepts and integrate it into a design methodology for self-optimizing systems. The present book summarizes the results of eleven years of research. It is one of two books that were created at the end of the CRC 614. The book “Dependability of Self-optimizing Mechatronic Systems” focuses on tools and methods to ensure the dependability of these systems during development and run-time.

The results of the CRC 614 represent a major milestone on the way to intelligent technical systems. While you are reading the present book, we are in the midst of the development of concepts and methods in follow-on projects for example for the industry 4.0. Please contact us, if you have any questions or contributions to discuss. Furthermore we would like to invite you to our virtual exhibition and to explore our CRC 614 interactively (`www.sfb614.de/en`).

Your CRC-Team

Paderborn,
October 2013

Prof. Dr.-Ing. Jürgen Gausemeier
Speaker of the Collaborative Research Centre 614

# Acknowledgements

This book was written within the context of the Collaborative Research Center 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering", which has been funded by the Deutsche Forschungsgemeinschaft (DFG).

It is testimony for the exemplary interdisciplinary collaboration and the team spirit. In the name of all members of the CRC 614 we want to thank the members of the interest group self-optimization (IG SO) and the authors as well as the many not mentioned discussion partner, without whom the book would not have been. Special thanks to Mrs. Dipl.-Wirt.-Ing. Mareen Vaßholz, who was coordinating the development of the book, to the coordinator of the IG SO Mr. Dipl.-Inform. Oliver Sudmann as well as to Dr. Thomas Ditzinger and his team from Springer-Verlag.

# Contents

**1 The Paradigm of Self-optimization** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.1 From Mechatronics to Intelligent Technical Systems . . . . . . . . . . . 2
1.2 Introduction to Self-optimization . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.3 Architecture of Self-optimizing Systems . . . . . . . . . . . . . . . . . . . . 8
1.3.1 Structure of Self-optimizing Mechatronic Systems . . . . . 8
1.3.2 Operator-Controller-Module . . . . . . . . . . . . . . . . . . . . . . . 10
1.4 Self-optimization in Intelligent Technical Systems . . . . . . . . . . . . . 12
1.4.1 Model-Based Self-optimization . . . . . . . . . . . . . . . . . . . . 12
1.4.2 Behavior-Oriented Self-optimization . . . . . . . . . . . . . . . . 16
1.4.3 Self-optimization by Reconfiguration . . . . . . . . . . . . . . . . 19
1.5 Structure of This Book . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

**2 Examples of Self-optimizing Systems** . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.1 Rail Technology – RailCab . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.1.1 Self-optimizing Operating Point Control . . . . . . . . . . . . . 29
2.1.2 intelligent Drive Module (iDM) . . . . . . . . . . . . . . . . . . . . 33
2.1.3 Active Guidance Module . . . . . . . . . . . . . . . . . . . . . . . . . 36
2.1.4 Active Suspension Module . . . . . . . . . . . . . . . . . . . . . . . . 38
2.1.5 Hybrid Energy Storage System (HES) . . . . . . . . . . . . . . . 42
2.1.6 Crosslinked Test Benches . . . . . . . . . . . . . . . . . . . . . . . . . 46
2.1.7 Convoy Mode . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
2.2 Miniature Robot BeBot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.2.1 Basic Vehicle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.2.2 Extension Modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
2.2.3 Operating System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
2.2.4 Implementing Self-X Properties . . . . . . . . . . . . . . . . . . . . 54
2.3 X-by-Wire Test Vehicle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
2.3.1 Vehicle Dynamics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
2.3.2 Self-optimizing Integrated Vehicle-Dynamics Control . . 58
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

**3 Development of Self-optimizing Systems** .......... 65
3.1 Design Methodology for Self-optimizing Systems .......... 66
3.2 Domain-Spanning Conceptual Design .......... 69
3.3 Domain-Specific Design and Development .......... 74
3.3.1 Mechanical Engineering .......... 76
3.3.2 Control Engineering .......... 78
3.3.3 Software Development .......... 82
3.3.4 Electrical and Electronic Engineering .......... 88
3.3.5 (Sub)system Integration .......... 91
3.3.6 Interaction of the Domains in the Design and Development .......... 93
3.4 Self-optimizing Development Process .......... 96
3.4.1 Framework of a Self-optimizing Development Process .......... 96
3.4.2 Systematic Planning of Synchronizations .......... 105
References .......... 113

**4 Methods for the Domain-Spanning Conceptual Design** .......... 117
4.1 Specification Technique CONSENS for the Description of Self-optimizing Systems .......... 119
4.2 Software Support for the Specification of the Principle Solution .......... 127
4.3 Consistency Analysis of Application Scenarios .......... 129
4.4 Design of the System of Objectives .......... 133
4.5 Design Framework for the Integration of Cognitive Functions Based on Solution Patterns .......... 139
4.5.1 Systems Analysis .......... 139
4.5.2 Functional Description .......... 141
4.5.3 Solution Selection .......... 143
4.5.4 Systems Specification .......... 147
4.6 Product Structuring for Self-optimizing Systems .......... 154
4.7 Early Probabilistic Reliability Analysis Based on the Principle Solution .......... 162
4.8 Evaluation of the Economic Efficiency .......... 171
References .......... 180

**5 Methods for the Design and Development** .......... 183
5.1 Automatic Model Transformation and Synchronization .......... 186
5.1.1 Example Scenario .......... 186
5.1.2 Deriving Initial Domain-Specific Models from the System Model .......... 188
5.1.3 Synchronizing Models during the Domain-Specific Refinement Phase .......... 193
5.2 Software Design .......... 197
5.2.1 Component Model .......... 198

5.2.2 Decompose Communication Requirements . . . . . . . . . . . 199
5.2.3 Real-Time Coordination Patterns . . . . . . . . . . . . . . . . . . . 200
5.2.4 Discrete Behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
5.2.5 Simulation of Hybrid Behavior . . . . . . . . . . . . . . . . . . . . . 220
5.2.6 Specification of Deployment . . . . . . . . . . . . . . . . . . . . . . . 221
5.2.7 Integration of Self-healing Behavior . . . . . . . . . . . . . . . . . 221
5.2.8 Code Generation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222
5.3 System Optimization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222
5.3.1 Set-Oriented Multiobjective Optimization . . . . . . . . . . . . 224
5.3.2 Hierarchical Multiobjective Optimization . . . . . . . . . . . . 229
5.3.3 Hierarchical Modeling of Mechatronic Systems . . . . . . . 232
5.3.4 Parametric Multiobjective Optimization . . . . . . . . . . . . . . 236
5.3.5 Computation of Robust Pareto Points . . . . . . . . . . . . . . . . 237
5.3.6 Optimal Control of Mechanical and Mechatronic Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239
5.3.7 Motion Planning with Motion Primitives . . . . . . . . . . . . . 244
5.3.8 Hierarchical Hybrid Planning . . . . . . . . . . . . . . . . . . . . . . 246
5.3.9 Statistical Planning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 250
5.3.10 Behavior Planning in Nondeterministic Environment . . . 253
5.3.11 FIPA Conform Cross-Domain Communication . . . . . . . . 257
5.3.12 Preparing Solution Pattern "Hybrid Planning" . . . . . . . . . 261
5.4 Dynamic Reconfiguration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 265
5.4.1 Fine-Grained Reconfigurable Architectures . . . . . . . . . . . 265
5.4.2 Coarse-Grained Reconfigurable Architectures . . . . . . . . . 267
5.4.3 Modelling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 267
5.4.4 Design Methods for Dynamic Reconfigurable Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 271
5.4.5 Platforms and Applications . . . . . . . . . . . . . . . . . . . . . . . . 279
5.5 System Software . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 291
5.5.1 Architecture for Self-optimizing Operating Systems . . . . 293
5.5.2 Self-optimized Flexible Resource Management . . . . . . . 297
5.5.3 Self-optimization in the Operating System . . . . . . . . . . . . 299
5.5.4 Hierarchical Flexible Resource Manager . . . . . . . . . . . . . 305
5.6 Virtual Prototyping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 310
5.6.1 Virtual Prototypes and Virtual Environments . . . . . . . . . . 311
5.6.2 Automatic Model Linking . . . . . . . . . . . . . . . . . . . . . . . . . 313
5.6.3 Visualization Agents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 320
5.6.4 Virtual Test Bench . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 330
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 337

**6 Summary and Outlook** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 351

**Index** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 353

# List of Contributors

Harald Anacker
Fraunhofer Institute for Production Technology IPT, Project Group Mechatronic Systems Design, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: `harald.anacker@ipt.fraunhofer.de`

Prof. Dr.-Ing. Joachim Böcker
Power Electronics and Electrical Drives, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: `boecker@lea.upb.de`

Christian Brenner
Heinz Nixdorf Institute, University of Paderborn, Software Engineering Group, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: `cbr@uni-paderborn.de`

Prof. Dr.-Ing. habil. Wilhelm Dangelmaier
Heinz Nixdorf Institute, University of Paderborn, Business Computing, especially CIM, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: `wilhelm.dangelmaier@hni.uni-paderborn.de`

Prof. Dr. Michael Dellnitz
Chair of Applied Mathematics, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: `dellnitz@uni-paderborn.de`

Rafal Dorociak
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: `rafal.dorociak@hni.uni-paderborn.de`

Dr.-Ing. Roman Dumitrescu
Fraunhofer Institute for Production Technology IPT, Project Group Mechatronic Systems Design, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: `roman.dumitrescu@ipt.fraunhofer.de`

Kathrin Flaßkamp
Computational Dynamics and Optimal Control, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: kathrinf@math.uni-paderborn.de

Tobias Gaukstern
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: tobias.gaukstern@uni-paderborn.de

Prof. Dr.-Ing. Juergen Gausemeier
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: juergen.gausemeier@hni.uni-paderborn.de

Stefan Groesbrink
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: stefan.groesbrink@hni.uni-paderborn.de

Philip Hartmann
Heinz Nixdorf Institute, University of Paderborn, Business Computing, especially CIM, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: philip.hartmann@hni.uni-paderborn.de

Christian Hölscher
Design and Drive Technology, University of Paderborn, Pohlweg 47-49, 33098 Paderborn, Germany
e-mail: c.hoelscher@uni-paderborn.de

Christian Heinzemann
Heinz Nixdorf Institute, University of Paderborn, Software Engineering Group, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: c.heinzemann@uni-paderborn.de

Christian Horenkamp
Chair of Applied Mathematics, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: christian.horenkamp@math.upb.de

Peter Iwanek
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: peter.iwanek@hni.uni-paderborn.de

Alexander Jungmann
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: alexander.jungmann@c-lab.de

Jan Henning Keßler
Heinz Nixdorf Institute, University of Paderborn, Control Engineering and Mechatronics, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: `jan.henning.kessler@hni.uni-paderborn.de`

Sebastian Korf
Heinz Nixdorf Institute, University of Paderborn, System and Circuit Technology, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: `korf@hni.uni-paderborn.de`

Dr. Lisa Kleinjohann
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: `lisa@c-lab.de`

Dr. Bernd Kleinjohann
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: `bernd@c-lab.de`

Martin Krüger
Heinz Nixdorf Institute, University of Paderborn, Control Engineering and Mechatronics, Fuerstenallee 11, D-33102 Paderborn, Germany
e-mail: `kruemar@uni-paderborn.de`

Dr.-Ing. Mario Porrmann
Cornitronics and Sensor Systems, Center of Excellence Cognitive Interaction Technology, Bielefeld University,Universitätsstraße 21-23, 33615 Bielefeld, Germany
e-mail: `mporrmann@cit-ec.uni-bielefeld.de`

Jun.-Prof. Dr. Sina Ober-Blöbaum
Computational Dynamics and Optimal Control, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: `sinaob@math.uni-paderborn.de`

Dr. Simon Oberthuer
Design of Distributed Embedded Systems, Heinz Nixdorf Institute, University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: `oberthuer@uni-paderborn.de`

Claudia Priesterjahn
Heinz Nixdorf Institute, University of Paderborn, Software Engineering Group, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: `c.priesterjahn@uni-paderborn.de`

Prof. Dr. Franz-Josef Rammig
Heinz Nixdorf Institute, University Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: `franz@uni-paderborn.de`

Christoph Rasche
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: crasche@c-lab.de

Peter Reinold
Heinz Nixdorf Institute, University of Paderborn, Control Engineering and Mechatronics, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: peter.reinold@hni.upb.de

Jan Rieke
Heinz Nixdorf Institute, University of Paderborn, Software Engineering Group, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: jan.rieke@uni-paderborn.de

Maik Ringkamp
Applied Mathematics, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: ringkamp@math.upb.de

Dr.-Ing. Christoph Romaus
Power Electronics and Electrical Drives, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: romaus@lea.upb.de

Prof. Dr. Wilhelm Schäfer
Heinz Nixdorf Institute, University of Paderborn, Software Engineering Group, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: wilhelm@uni-paderborn.de

Thomas Schierbaum
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: thomas.schierbaum@hni.uni-paderborn.de

Christoph Schulte
Power Electronics and Electrical Drives, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: schulte@lea.upb.de

Christoph Sondermann-Woelke
Mechatronics and Dynamics, University of Paderborn, Pohlweg 47-49, 33098 Paderborn, Germany
e-mail: christoph.sondermann-woelke@uni-paderborn.de

Katharina Stahl
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: katharina.stahl@uni-paderborn.de

Dominik Steenken
Specification and Modelling of Software Systems, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: dominik@uni-paderborn.de

Karl Stephan Stille
Power Electronics and Electrical Drives, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: stille@lea.upb.de

Dr. Jörg Söcklein
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: Joerg.Stoecklein@hni.uni-paderborn.de

Oliver Sudmann
Heinz Nixdorf Institute, University of Paderborn, Software Engineering Group, Zukunftsmeile 1, 33102 Paderborn, Germany
e-mail: oliversu@uni-paderborn.de

Robert Timmermann
Applied Mathematics, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: robert.timmermann@math.upb.de

Prof. Dr.-Ing. habil. Ansgar Trächtler
Heinz Nixdorf Institute, University of Paderborn, Control Engineering and Mechatronics, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: ansgar.traechtler@hni.upb.de

Mareen Vaßholz
Heinz Nixdorf Institute, University of Paderborn, Product Engineering, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: mareen.vassholz@hni.uni-paderborn.de

Prof. Dr. Heike Wehrheim
Specification and Modelling of Software Systems, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: wehrheim@uni-paderborn.de

Dr. Katrin Witting
Applied Mathematics, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: witting@math.upb.de

Yuhong Zhao
Heinz Nixdorf Institute, University of Paderborn, Design of Distributed Embedded Systems, Fuerstenallee 11, 33102 Paderborn, Germany
e-mail: `zhao@uni-paderborn.de`

Steffen Ziegert
Specification and Modelling of Software Systems, University of Paderborn, Warburger Straße 100, 33098 Paderborn, Germany
e-mail: `steffen.ziegert@uni-paderborn.de`

Prof. Dr.-Ing. Detmar Zimmer
Design and Drive Technology, University of Paderborn, Pohlweg 47-49, 33098 Paderborn, Germany
e-mail: `Detmar.Zimmer@uni-paderborn.de`

# Chapter 1
# The Paradigm of Self-optimization

Michael Dellnitz, Roman Dumitrescu, Kathrin Flaßkamp, Jürgen Gausemeier, Philip Hartmann, Peter Iwanek, Sebastian Korf, Martin Krüger, Sina Ober-Blöbaum, Mario Porrmann, Claudia Priesterjahn, Katharina Stahl, Ansgar Trächtler, and Mareen Vaßholz

**Abstract.** Machines are ubiquitous. They produce, they transport. Machines facilitate and assist with work. The increasing fusion of mechanical engineering with information technology has brought about considerable benefits. This situation is expressed by the term mechatronics, which means the close interaction of mechanics, electrics/electronics, control engineering and software engineering to improve the behavior of a technical system. The integration of cognitive functions into mechatronic systems enables systems to have inherent partial intelligence. The behavior of these future systems is formed by the communication and cooperation of the intelligent system elements. From an information processing point of view, we consider these distributed systems to be multi-agent-systems. These capabilities open up fascinating prospects regarding the design of future technical systems. The term self-optimization characterizes this perspective: the endogenous adaptation of the system's objectives due to changing operational conditions. This resuls in an autonomous adjustment of system parameters or system structure and consequently of the system's behavior. In this chapter self-optimizing systems are described in detail. The long term aim of the Collaborative Research Centre 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" is to open up the active paradigm of self-optimization for mechanical engineering and to enable others to develop these systems. For this, developers have to face a number of challenges, e.g. the multidisciplinarity and the complexity of the system. This book povides a design methodology that helps to master these challenges and to enable third parties to develop self-optimizing systems by themselves.

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*,
Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_1, © Springer-Verlag Berlin Heidelberg 2014

## 1.1 From Mechatronics to Intelligent Technical Systems

Roman Dumitrescu, Jürgen Gausemeier, Peter Iwanek, and Mareen Vaßholz

Machines are ubiquitous, from white goods to medical devices and transportation systems. Their purpose is to make life easier. The increasing integration of information and communication technology in the field of conventional mechanical engineering implies considerable potential for innovation. Most modern products created in the field of mechanical engineering and related areas such as automobile technology, already rely on the close symbiotic interaction between mechanics, electrics/electronics, control engineering and software technology, which is aptly expressed by the term mechatronics [29] . Mechatronics – a portmanteau of the words mechanics and electronics [10] – represents the potential for the development of fundamentally new solutions in the area of mechanical engineering and related disciplines, which in turn can significantly improve the cost-benefit ratio of familiar products and also stimulate the innovation of new products. In the following, the way from Mechatronic Systems across Adaptive Systems towards Intelligent Technical Systems will be outlined.

### Mechatronic Systems

The aim of mechatronics is to improve the behavior of a technical system by using sensors and information from humans or other systems to obtain information on the environment and also on the system. Thus, they can respond to changes in their environment, detect critical operating states and control processes which are difficult to control in real-time by humans [2, 22, 34]. Mechatronic systems are distinguished by the functional and/or spatial integration of sensors, actuators, information processing and a basic system. The basic structure of a mechatronic system is shown in Fig. 1.1. In general, mechatronic systems can also be composed of subsystems which themselves are mechatronic systems (cf. Sect. 1.3) [2].

The **basic system** is commonly a mechanical structure. Generally any desired physical system is conceivable as a basic system, so that it is even possible to hierarchically represent mechatronic systems structurally (cf. Sect. 1.3) [2]. The relevant physical (continuous) values of the basic system or its environment are measured using **sensors**. Sensors in this case can be physically present measured-value pickups or straightforward software sensors [2] (so-called "observers", see for example [34]). The sensors supply the input variables for **information processing**, which in most cases takes place digitally, i.e. discretely in terms of value and time. The information processing unit determines the necessary changes to the basic system using the measurement data as well as the user specifications (human-machine interface) and also available information from other processing units (communication system). The information processing unit is often realized as an electronic microprocessor, which realize open-loop and closed-loop control functions. The behavior adaptation of the basic system will be caused by **actuators**. In general, an

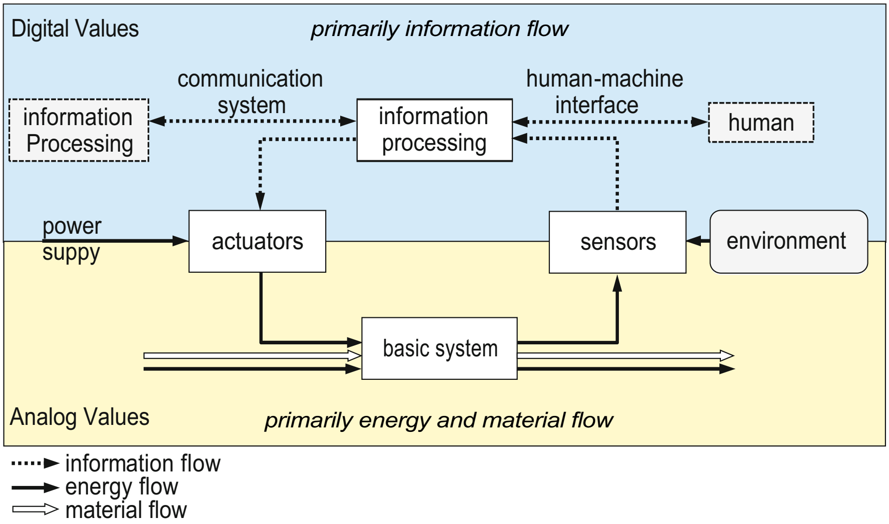


**Fig. 1.1** Basic structure of a mechatronic system [2]

actuator converts energy into motion, for example by using a motor or a hydraulic cylinder.

The relationships between the basic system, sensors, information processing and actuators are represented as flows (cf. Fig. 1.1). For example there is a flow from the sensors (e.g. informations of the environment) to the information processing unit of the system. In principle, three types of flows can be distinguished: information flows, energy flows and material flows [44], whereby the information flows are frequently also referred to as signal flows.

- **Material flows:** Examples for materials which flow between units of mechatronic systems are for example gases, fluids, solids, dust and also raw products, objects being tested or objects being treated.
- **Energy flows:** Energy is understood to be any form of energy, such as for example mechanical, thermal, electrical, optical energy, or force and electric current.
- **Information flows:** Information exchanged between units of mechatronic systems or components are, for example measured variables, control pulses, or data.

To serve customer needs and to create cost and energy efficient systems, mechatronic systems have to fulfill increasingly advanced functions and requirements. These requirements can vary or conflict depending on the situation. An example is the consideration of execution speed and resource use: if the application situation requires the fastest possible execution of a task (e.g. an important rush order), the requirement resource efficiency is considered to be less important. In the development of conventional mechatronic systems, the developer has to decide on a compromise between two competing requirements. Thus, the controller of the system is designed for acceptable behavior under specific circumstances. A first step to solve this challenge is to use adaptive control strategies [3, 8].

### Adaptive Systems

Adaptive systems use adaptive control strategies, that make it possible to adjust the controller of the system in real time by using **adaptation algorithms**. These algorithms help to achieve a desired level of control system performance for situations in which the parameters of the basic system are a priori unknown and/or change in time [39]. Therefore adaptive systems form a necessary step towards intelligent technical systems. Adaptive controllers enable higher levels of adaptability, but still remain a compromise for anticipated situations [3]. The improvement of communication and information technology opens up more and more fascinating perspectives, which go far beyond current standards of mechatronics: mechatronic systems having inherent partial intelligence.

### Intelligent Technical Systems

Future mechanical engineering systems will comprise of configurations of system elements with inherent partial intelligence. These system elements are able to realize functions such as "to share knowledge", "to coordinate behavior" or "to learn from experience". Those functions are also typical for cognitive systems and are known as cognitive functions [16]. Systems having these cognitive functions, are also called intelligent technical systems. There are four central properties, which describe intelligent technical systems [14, 15]:

- **Adaptive:** Adaptive systems react autonomously and flexibly on changing operation conditions. They are able to learn and optimize their behavior at runtime.
- **Robust:** These systems are able to behave "acceptable", even in situations which have not been considered during the development phase of the system. Uncertainties can be compensated to certain extends.
- **Foresighted:** Based on gained experience, these systems have the ability to recognize emerging states and situations. Thus they are able, to spot possible dangers and accordingly change their behavior.
- **User-friendly:** The systems are able to adapt their behavior to the specific user. They interact closely with the user and their behavior is always comprehensible.

Keywords as "Things that Think", "Cyber-Physical Systems", "Industry 4.0" or "Self-optimization" express this perspective on intelligent technical systems. These systems exceed the functionality of mechatronic systems and set new requirements on design methodologies. Therefore the gap between system complexity and performance of the design methodology is increasing (see Fig. 1.2). To realize intelligent technical systems, such as self-optimizing systems, not only the domains mechanical, software, control and electrical/electronic engineering have to be considered, but also experts from higher mathematics and artificial intelligence have to

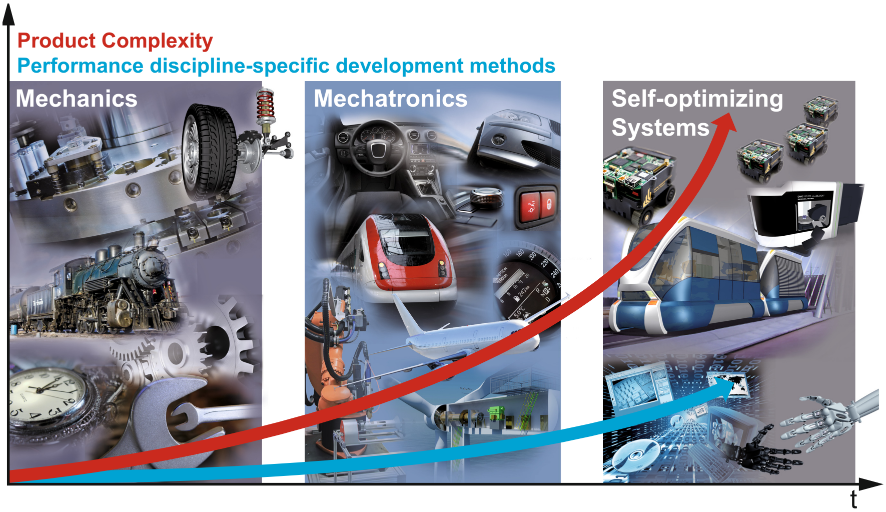


**Fig. 1.2** Complexity of self-optimizing systems versus performance of design methodologies

be involved in the development process. This book provides a design methodology for self-optimizing systems consisting of a reference process, methods and tools that master the shortcomings of existing design methodologies. This methodology closes the gap between system complexity and performance of the design methodology, to enable developers who are not specifically trained in higher mathematics and artificial intelligence to develop self-optimizing systems.

## 1.2 Introduction to Self-optimization

Roman Dumitrescu, Jürgen Gausemeier, and Peter Iwanek

The Collaborative Research Center (CRC) ’Self-Optimizing Concepts and Structures in Mechanical Engineering’ defines **self-optimization** as follows:

> *"Self-optimization describes the ability of a technical system to endogenously adapt its objective regarding changing influences and thus adapt the system’s behavior in accordance to the objectives. The behavior adaptation may be performed by changing the parameters or the structure of the system. Thus self-optimization goes considerably beyond the familiar rule-based and adaptive control strategies; Self-optimization facilitates systems with inherent "intelligence" that are able to take action and react autonomously and flexibly to changing operating conditions." [3]*

Figure 1.3 shows the key aspects and the mode of operation of a self-optimizing system. Factors that **influence the technical system** originate in its surroundings (environment, users, etc.) or from the system itself. They can support the system’s

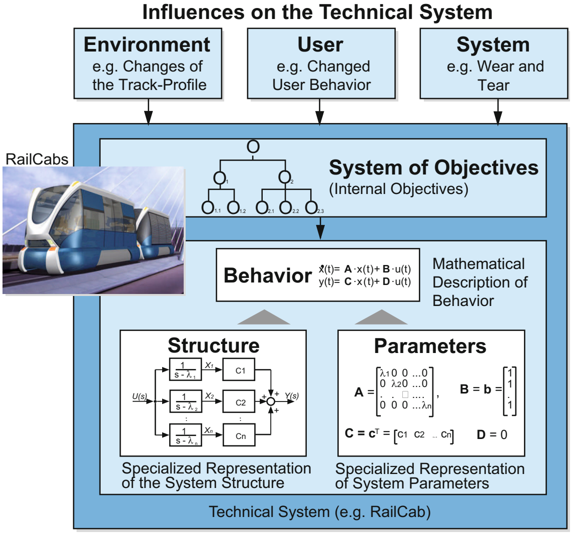

Influences on the Technical System
Environment
e.g. Changes of the Track-Profile
User
e.g. Changed User Behavior
System
e.g. Wear and Tear
RailCabs
O
O1
O2
O1.1
O1.2
O2.1
O2.2
O2.3
System of Objectives
(Internal Objectives)
Behavior
ẋ(t)= A·x(t)+ B·u(t)
y(t)= C·x(t)+ D·u(t)
Mathematical Description of Behavior
Structure
U(s)
1/(s - λ1)
1/(s - λ2)
1/(s - λn)
X1
X2
Xn
C1
C2
Cn
+
+
+
Y(s)
Parameters
A = [λ1 0 0 ...0; 0 λ2 0 ...0; . . ....; 0 0 0 ...λn],
B = b = [1; 1; .; 1]
C = cT = [c1 c2 ... cn]
D = 0
Specialized Representation of the System Structure
Specialized Representation of System Parameters
Technical System (e.g. RailCab)

**Fig. 1.3** Aspects of self-optimizing systems [21, 25]

objectives or hinder them. Influences from the **environment**, for example such as strong winds or icy conditions, are unstructured and often unpredictable. If they hinder the system fulfilling its pursued objectives they are called disturbance variables. The **user** can influence the system, for instance by choosing preferred objectives. It is also possible that the **system** itself or other technical systems will influence the system's objectives, for example if mechanical components are damaged, the objective "max. safety" has to be prioritized [20].

The self-optimizing system determines its current pursued objectives (**System of Objectives**) on the basis of the encountered influences on the system for example the environment, the user or the system itself. New objectives can be added, existing objectives can either be rejected or the priority of objectives can be modified during system operations. Therefore, the system of objectives and its autonomous change is the core of self-optimization [3, 15]. Objectives can be distinguished between external and inherent objectives. External objectives are set from the outside of the self-optimizing system, by other systems or by the user (e.g. for a driving module this could be "max. comfort"). Inherent objectives reflect the design purpose of the self-optimizing system. An inherent objective of a driving module can be for example "max. energy efficiency". Objectives build a hierarchy and each objective can thus be refined by sub-objectives (e.g. "min. energy consumption" is a possible sub-objective of "max. energy efficiency"). Inherent and external objectives that are pursued by the system at a given moment during its operation are called internal objectives [13].

Adapting the objectives, leads to a continuous adjustment of the **system behavior** to the occuring situation. This is achieved by adapting **parameters** or reconfiguring the **structure** (e.g. adapting control strategies) [15]. The self-optimization process consists of the following three actions [23]:

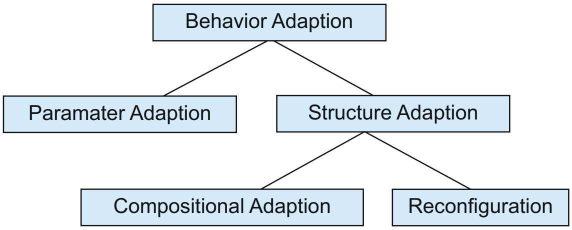


**Fig. 1.4** Behavioral adaptation in a self-optimizing system by structure and/or parameter adaptation [3]

1. **Analyzing the current situation:** The current situation includes the current state of the system as well as all observations of the environment that have been carried out. Observations can also be made indirectly by communication with other systems. Furthermore, a system's state contains previous observations that were saved. One basic aspect of this first step is the analysis of the fulfillment of the objectives [3].

2. **Determining the system's objectives:** The system's objectives can be extracted by choice, adjustment, and generation. By choice means the selection of one alternative output of a predetermined quantity of possible objectives; the adjustment of objectives means the gradual modification of existing objectives respective to their relative weighting. Generation means, if new objectives are being created that are independent from the existing ones [3].

3. **Adapting the system behavior:** The changed system of objectives demands an adaptation of the behavior of the system and its components. As mentioned before, this can be realized by adapting the parameters and, if required, by adapting the structure of the system. The different types of behavior adaptation strategies are shown in Fig. 1.4. Parameter adaptation means for example changing a control parameter. Structure adaptations affect the arrangement of the system elements and their relationships. Here we distinguish between reconfiguration, which changes the relationships between a fixed set of available elements, and compositional adaptation, in which new elements are integrated into the existing structure or existing elements are removed from it [20]. The self-optimization process leads, according to changing influences, to a new system state. Thus a state transition takes place [3]. The behavior adaptation finally concludes the self-optimization process.

The self-optimization process takes place if the three actions are performed repeatedly by the system. The three actions do not need to be performed in a specified sequence. For example, within the scope of planning, different situations are considered and according to the situations, the objectives are adapted. This results in repeated situation analysis, based on the determination of objectives. Thus, the self-optimization process is executed, if the situation of the system is changed or a planning for possible system scenarios is performed [3].

Thus, self-optimization can be considered as an extension of classical and advanced control engineering [8]. In order to provide an optimal conformity to the environment of the system at any time, self-optimizing systems utilize implemented adaptation strategies, instead.

To control self-optimizing systems, a consistent structuring of the information processing is needed. We distinguish between the macro structure of the mechatronic system and the structure of the information processing, represented by the Operator-Controller-Module. These structures will be explained in detail in the following section.

## 1.3 Architecture of Self-optimizing Systems

Martin Krüger and Ansgar Trächtler

Two types of structuring are presented. First, a description is given on how the entire system can be divided into subsystems or modules according to their function within the system. Such a decoupling naturally leads to a hierarchical ordering of the modules, with simpler modules on the lowest level and the entire system on the topmost level [40]. Second, the Operator-Controller-Module (OCM), a multi-level architecture, is introduced [32]. It includes all of the types of information processing which are necessary to realize an intelligent system: classical quasi-continuous controllers, discrete or event-based methods like error analysis and monitoring concepts as well as methods for cognitive capabilities, e.g. learning or optimization algorithms, to name but a few. Both structuring types complement one another and can be used in combination.

### *1.3.1 Structure of Self-optimizing Mechatronic Systems*

One main step in the design of self-optimizing systems is to develop a hierarchy of functions based on the system requirements, see 4.1 for more details. A self-optimizing system can be divided into subsystems using this hierarchy of functions. The first step is to create a hierarchy of *motion functions* which describe the controlled motion of bodies, c.f. [3, 31]. Each motion function of the hierarchy can be realized by one of the three structuring elements:

**Mechatronic Function Module (MFM):** The MFMs are the basic elements of the entire mechatronic system. Each MFM includes sensors, actuators, information processors and the basic mechanical system. The motion of the mechanical system is measured using sensors and can be controlled by means of actuators. The control input is computed by the information processing. The actuators of a MFM can again be given by another MFM.

**Autonomous Mechatronic System (AMS):** The AMS is on the top level of an actual mechatronic system. It is associated with the complete mechanical structure of the physical element and thus forms the top level of the mechanical structure. Besides the associated mechanical structure, the AMS includes sensors and information processing elements. There is no need for actuators in an AMS. The actuating elements are given by the underlying MFMs and are coordinated by the AMS.

**Networked Mechatronic System (NMS):** NMS elements make up the top level of the hierarchy. A NMS is comprised of information processing and sensors. However, the CNS is made up of several AMS, which are linked by signals alone. The NMSs do not necessarily have their own physical representation in terms of a data-processing hardware. The function of the NMS might be implemented in the AMS, so that the NMS is generated whenever several AMS are interconnected.

Figure 1.5 exemplarily shows the hierarchical structure of the RailCab System which is described in detail in Sect. 2.1. On the topmost level a convoy consisting of a couple of RailCabs is represented by a NMS. Each RailCab itself is described by an AMS and includes several MFMs, e.g. the active suspension system. The information processing units of the structuring elements can be seen as agents. An agent in this context is an information processing unit which is used to fulfill a particular functionality pursuing the corresponding objectives. It analyzes its environment conditions and has the ability to adjust its own behavior autonomously according to the current situation and to the requirements of the remaining agents. In this way hierarchically structured self-optimizing systems can be seen as multi-agent systems.

Information processing of each structuring element can itself be a complex unit consisting of several software components. Hence, it also has to be structured in

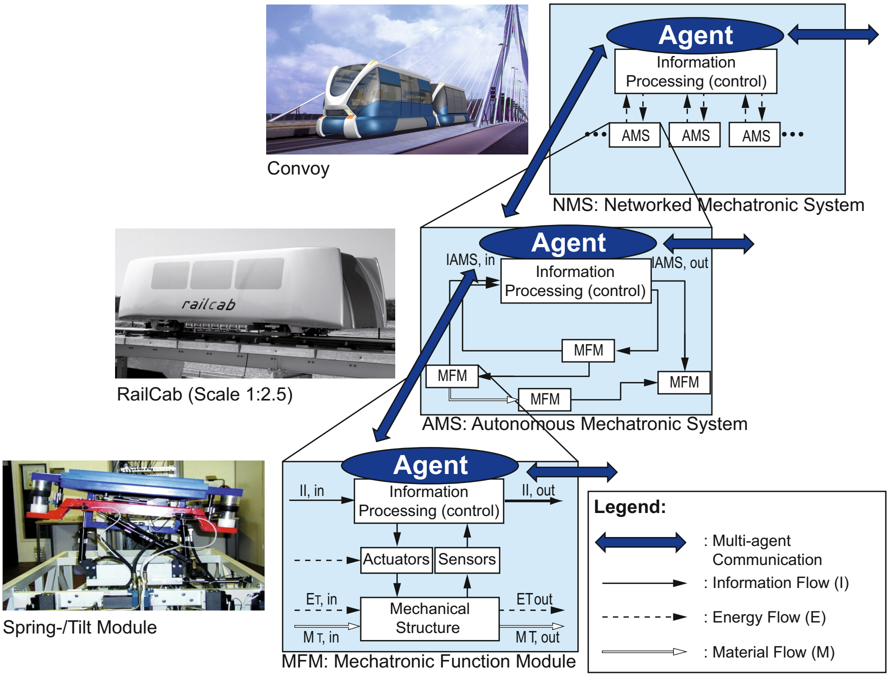


**Fig. 1.5** Structure of intelligent mechatronic systems [3]

order to ensure a systematic design and a dependable functionality. The Operator-Controller-Module described in the following section can be used to structure the information processing of self-optimizing systems.

### *1.3.2 Operator-Controller-Module*

The information processing unit of a self-optimizing system has to perform a multitude of functions: quasi-continuous control of the plant motion, monitoring in view of occurring malfunctions, adaptation of the control strategy to react to changing environmental conditions, communication with other systems to name a few of these functions.

In order to ensure a clear and manageable information processing, an architecture is needed which contains all these functions. Additionally, the hierarchical structuring concept described in the last section has to be taken into account. The **Operator-Controller-Module** (OCM) is an architecture with three levels that has been proven to be an advantageous and effective structure for self-optimizing systems (see Fig. 1.6). It is based on results of cognition science, see [48], and was first published in [32]. It is used for the information processing on each level of a complex mechatronic system. The result is a hierarchy of OCMs that is also beneficial for modeling and optimization as described in detail in Sec. 5.3.3. The three different levels of an OCM are geared to the kind of effect on the technical system.

**Controller:** The controller, which is on the lowest level, realizes the desired dynamical behavior of the plant. It is similar to a classical control loop. Measurements are used to compute control signals which directly affect the plant. Hence, it can be called a “Motor Loop”. The controller operates in a quasi-continuous way under hard real-time constraints. Several types of controllers can be implemented at the same time with the possibility to switch between them. Different switching strategies can be used, e.g. a flatness-based approach presented in [42].

**Reflective Operator:** The reflective operator monitors and regulates the controller. It consists of sequential control, emergency routines as well as adaptation algorithms for the control strategies. The reflective operator does not access the actuators of the system directly, but modifies the controller by initiating changes of controller parameters or switching between different controllers. The reflective operator works mostly in an event-oriented manner. It also has to operate under hard real-time constraints, because it is tightly linked to the controller. However, it is also the connecting element to the cognitive level of the OCM and provides an interface between those elements that are not capable to operate in real-time and the controller. It filters incoming signals and results from the cognitive level and inputs them to the subordinated level.

**Cognitive Operator:** The topmost level of the OCM is represented by the cognitive operator. On this level the system can gather information on itself and its environment by applying various methods such as learning, use of knowledge-based systems, or model-based optimization. The results can be used to improve the

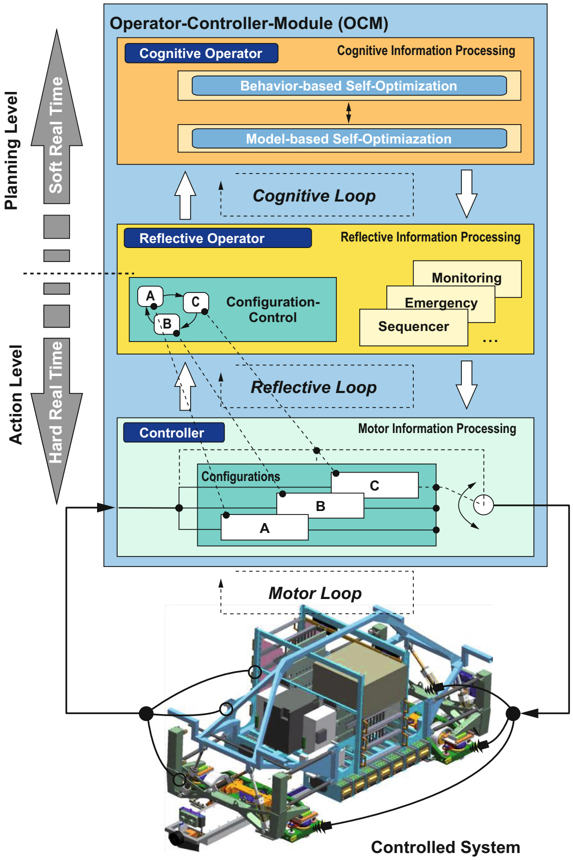


**Fig. 1.6** Structure of Operator-Controller-Module [3]

system behavior. This optimizing information processing can roughly be divided into model-based and behavior-oriented optimization, introduced in Sect. 1.4.1 and Sect. 1.4.2, respectively. The former class of optimization techniques is based on a model for the dynamical behavior of technical systems while the latter uses methods from artificial intelligence and soft-computing. While both the controller and the reflective operator are subject to hard real-time constraints, the cognitive operator can also operate asynchronously to real-time. Nevertheless, it has to respond within a certain time limit. Otherwise, self-optimization would not find utilizable results in view of changing environmental conditions. Hence, the cognitive operator is subject to soft real-time. Consequently we are able to integrate cognitive functions into the technical system that previously only biological systems were capable of.

## 1.4 Self-optimization in Intelligent Technical Systems

Michael Dellnitz, Kathrin Flaßkamp, Philip Hartmann, and Sina Ober-Blöbaum

Self-optimizing systems adapt their behavior according to current situations and objectives. Therefore, appropriate strategies and methods have to be implemented into the Cognitive Operator of the Operator-Controller-Module (cf. Fig. 1.6). As introduced in Sect. 1.2 (cf. in particular Fig. 1.4), the system's adaptation can be realized by parameter adaption and/or by reconfiguration. Finding parameters that optimize a current set of objectives is an optimization or an optimal control problem. Methods to solve these kind of problems typically rely on models of the system's dynamic behavior. In Sect. 1.4.1, an introduction to formal problem statements and solution approaches is given for these *model-based methods for self-optimization*. In case an explicit physical model of the system or process is not available, *behavior-oriented self-optimization* is used. These approaches work on a mapping of input values to output values (cf. Sect. 1.4.2). Strutural reconfiguration affects all levels of a self-optimizing system, from software to hardware. This kind of adaptation is realized by exchanging system parts, e.g. software components or areas of FPGAs (Field Programmable Gate Arrays). We will provide a closer look on reconfiguration in Sect. 1.4.3.

### *1.4.1 Model-Based Self-optimization*

Michael Dellnitz, Kathrin Flaßkamp, and Sina Ober-Blöbaum

The development of self-optimizing mechatronic systems requires the solution of optimization problems from the early design phase to system operation. Model-based design techniques, which are state of the art in particular in control engineering, allow an automatic, model-based computation of solutions that are guaranteed to be optimal for the given problems by numerical optimization methods.

Optimization problems are classified by the type of variables, which can be discrete or continuous. **Discrete optimization problems** typically arise in logistic or planning problems where long term forecasts have to be computed and can be either addressed with discrete model-based or discrete behavior-based methods. The optimization of the design and the dynamical behavior of intelligent mechatronic systems gives rise to various **continuous optimization problems**. If time-dependent steering maneuvers for technical systems or processes have to be optimized, we are faced with **optimal control problems**. In many applications, in particular for self-optimizing systems, there are several objectives which have to be simultaneously optimized leading to **multiobjective optimization**. Regarding the process of self-optimization, it is multiobjective optimization which enables the identification of objectives (step 2 of the self-optimization process, cf. Sect. 1.2) during operation.

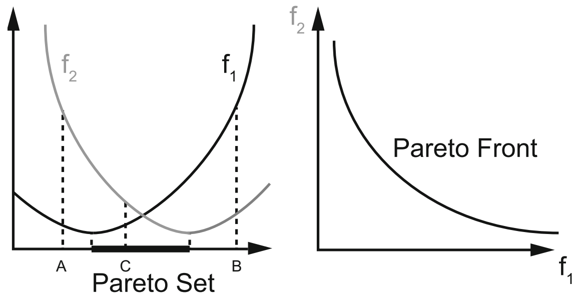


**Fig. 1.7** Sketch of an MOP with two objectives $f_1$ and $f_2$. While the points (A) and (B) are not optimal, (C) is a point of the Pareto set

### 1.4.1.1 Multiobjective Optimization

Multiobjective optimization (sometimes also called multicriteria optimization) takes several conflicting objectives into account and searches for optimal compromises, so called **Pareto points**. Simple examples of trade-offs between conflicting objectives are "minimal energy consumption versus minimal time" or "maximal quality, but minimal time and minimal (e.g. production) costs". A number of detailed examples of concurring objectives in technical applications are given in Sect. 2.

While ordinary optimization problems typically have a single global optimum, the solution of multiobjective optimization problems results in an entire set of Pareto points, the **Pareto set**. Formally, the multiobjective optimization problem (MOP) is stated as

$$\min\{\mathbf{F}(\mathbf{p}) : \mathbf{p} \in \mathbb{R}^n\}, \tag{1.1}$$

with $\mathbf{F}$ being a vector of objectives $f_1,\dots,f_k : \mathbb{R}^n \to \mathbb{R}$, i.e. $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^k$, $\mathbf{F}(\mathbf{p}) = (f_1(\mathbf{p}),\dots,f_k(\mathbf{p}))$. Here, $\mathbf{p}$ denotes the optimization parameters. These could be design parameters for the mechanical or electrical subsystem, for instance, or control parameters of the regulators. Minimization is meant with respect to the following partial ordering $\leq_p$ on $\mathbb{R}^n$: given $\mathbf{u},\mathbf{v} \in \mathbb{R}^n$, the vector $\mathbf{u}$ is smaller than the vector $\mathbf{v}$, $\mathbf{u} \leq_p \mathbf{v}$, if $u_i \leq v_i$ for all $i \in \{1,\dots,k\}$. The solutions of MOP can then be defined as follows: a point $\mathbf{p}^\star \in \mathbb{R}^n$ is called globally Pareto optimal for MOP (or a global Pareto point for MOP)if there does not exist any $\mathbf{p} \in \mathbb{R}^n$ with $\mathbf{F}(\mathbf{p}) \leq_p \mathbf{F}(\mathbf{p}^\star)$ and $f_j(\mathbf{p}) < f_j(\mathbf{p}^\star)$ for at least one $j \in \{1,\dots,k\}$. That means, no other point $\mathbf{p}$ gives a better or equal (but not entirely identical) value in all objectives. However, there typically exists other Pareto optimal points which are, compared to $\mathbf{p}^\star$, better in one objective but worse in another. Figure 1.7 gives an illustration of an MOP and Pareto optimal points. If the Pareto optimality property only holds for some neighborhood $U(\mathbf{p}^\star) \subset \mathbb{R}^n$, $\mathbf{p}^\star$ is called locally Pareto optimal. The image of the Pareto set, i.e. the corresponding function values, is called the Pareto front (cf. Fig. 1.7).

Necessary optimality conditions for Pareto optimality are given by the Karush-Kuhn-Tucker (KKT) equations, i.e. for an optimal point $\mathbf{x}^\star$, there exist multipliers $\beta_i \in \mathbb{R}$ with $\beta_i \geq 0$ and $\sum_{i=1}^{k} \beta_i = 1$ such that

$$\sum_{i=1}^{k} \beta_i \nabla f_i(\mathbf{p}^\star) = 0. \tag{1.2}$$

The MOP (cf. Eq. (1.1)) can be extended to include equality or inequality constraints, which have to be considered in the KKT equations involving additional terms with additional multipliers. Iteratively solving Eq. (1.2) to determine Pareto points $\mathbf{p}^\star$ is the basic idea of many multiobjective optimization algorithms.

For the solution of real world multiobjective optimization problems, numerical techniques have to be applied. There exist a number of methods for the computation of single Pareto points, for an overview we refer to [17]. However, for self-optimizing systems, it is important to gather knowledge about the entire Pareto set for later selections of specific design configurations, the **decision making** (cf. Sect. 1.4.1.3) during operation of the system. In the last decades, a number of techniques for the computation of entire Pareto sets have been developed (cf. e.g. [9, 11, 33, 37, 46]). In the course of the research of the CRC 614, set-oriented methods for multiobjective optimization (cf. e.g. [12] for an early reference or [47] for an overview) have been developed. Due to the approximation of the entire Pareto set (or the front, respectively) by box coverings, the methods are outstanding in their robustness and applicability to real world MOP problems, in particular for self-optimizing systems. These techniques are described in detail in Sect. 5.3.1 (cf. also Sect. 5.3.2, Sect. 5.3.4, and Sect. 5.3.5 for extensions to hierarchical and parametric multiobjective optimization problems).

In the following, we give a short introduction to optimal control problems, which often arise in control applications for technical systems. Solution methods for problems with single and multiple objectives are presented in Sect. 5.3.6.

### 1.4.1.2 Optimal Control

Optimal control problems arise, when the system's dynamical behavior has to be optimized by determining a time-dependent steering maneuver. In other words, such an optimal maneuver has to satisfy certain constraints and has to minimize a given cost functional like the control effort or the maneuver time. Typical examples are the optimal control of open chain industrial robots, the finding of optimal paths for vehicles, or the optimal control of engines. Formally, an optimal control problem (OCP) is defined by a cost functional (1.3a), e.g. the control effort, the time duration, or the deviation to a reference path, that has to be minimized with respect to several constraints:

$$\min_{\mathbf{x}(t),\mathbf{u}(t)} J(\mathbf{x},\mathbf{u}) = \int_0^T C(\mathbf{x}(t),\mathbf{u}(t))\,dt \tag{1.3a}$$

$$\text{with respect to } \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) \tag{1.3b}$$

$$\mathbf{r}(\mathbf{x}(0),\mathbf{x}(T)) = 0, \text{ and} \tag{1.3c}$$

$$\mathbf{h}(\mathbf{x}(t),\mathbf{u}(t)) \leq 0. \tag{1.3d}$$

The dynamical system (1.3b) describes the system's equations of motion in its state $\mathbf{x}$ under the influence of some control $\mathbf{u}$. Equations (1.3c) and (1.3d) are called boundary and path constraints, respectively, and take into account technical restrictions on the states or controls.

There exists a number of different approaches for numerically solving single objective OCP, for a good overview we recommend [7] and the references therein. The solution methods can be divided into indirect and direct methods. While indirect methods generate and then solve a boundary value problem according to the necessary optimality conditions of the Pontryagin maximum principle[1], direct methods start with a discretization of the problem (1.3a)-(1.3d). Thus, one obtains a nonlinear optimization problem that can be addressed by appropriate state of the art techniques such as sequential quadratic programming (SQP, cf. e.g. [28]).

In the case of differentially flat systems, the entire dynamics of the technical system can be described via (artificial) outputs and therefore, only the output functions have to be approximated by a finite number of parameters (equally spread nodes or splines as in [26]). Otherwise, the system's continuous states have to be discretized (by a finite number of nodes or even by some appropriately chosen short pieces of trajectories, so called primitives, cf. Sect. 5.3.7). A method that is especially tailored to the optimal control of mechanical systems (*DMOC, Discrete Mechanics and Optimal Control*, cf. [41]) is presented in Sect. 5.3.6.

If several cost functionals of the form (1.3a) have to be optimized simultaneously (e.g. the energetic effort and the duration of a steering maneuver), we are faced with a **multiobjective optimal control problem**. In Sect. 5.3.6, a method is presented that combines a multiobjective optimization algorithm with a direct optimal control technique to address problems of this kind.

#### 1.4.1.3 Decision Making and Self-optimization

The computation of entire Pareto sets of multiobjective optimization or optimal control problems is computationally costly but important for the design of a knowledge base on which the self-optimization during operation time relies. The Pareto optimal alternatives are computed offline in advance and stored in this knowledge base. During operation, one specific optimal configuration of the Pareto set has to be chosen at every time: this process is called decision making.

If only a (small) finite number of Pareto points is stored in the knowledge base, one possibility to implement the decision making process is given by the **hybrid planning** method, cf. Sect. 1.4.2 below and Sect. 5.3.8. Self-optimization based on precomputed knowledge bases of Pareto sets can be also realized by path following methods for parameter-dependent MOPs (cf. Sect. 5.3.4 and [49] for details) or in a model predictive control fashion for scalarized online optimization problems (cf. [26]).

[1] The Pontryagin maximum principle and a discussion of indirect optimal control methods are e.g. given in [7].

### *1.4.2 Behavior-Oriented Self-optimization*

Philip Hartmann

The term behavior-oriented optimization describes methods without an explicit physical model of the system or process. Instead, these approaches work on mapping input values to output values. The actual system and the considered process are observed as a black box and usually a discretization of the processes goes hand in hand. The relationship between system state, behavior, and objectives are provided by the developers of the system or learned by the system by using learning methods and exploration strategies. Because a classification of the environmental conditions and the system behavior are assumed when using defaults defined by experts or learning methods, the model of the behavior-oriented self-optimization is in general coarser than the model of the model-based self-optimization (cf. Sect. 1.4.1). Thus it is possible to plan the system's behavior for longer planning horizons.

**Planning** refers to a process that determines and examines the future behavior of the system instead of considering only the current situation. To analyze alternative options for execution, planning methods work on simplified models of the system behavior. An integral part of the simplification is the mapping to a discrete state space, so that only a snapshot of the system state at defined points in time will be considered. In general this approach corresponds to the definition of the planning problem in artificial intelligence, which can be used with a variety of methods. Due to the exploration of the state space for future situations a proactive reaction to future influences or avoidance of undesirable situations is made possible by the planning [3].

Any task of a mechatronic system (e.g. the transportation of persons or goods between two locations) can be expressed by a function [43]. This function describes the relationship between input and output variables [1] of the system by converting incoming energy, material and information flows into outgoing flows of the same types. Subtasks (such as driving with an active suspension system) are represented by analogous partial functions which are logically connected and make up the hierarchy of the overall function. The effect of a partial function also depends on the physical effect leading to a particular partial function solution [43]. Thus, partial functions can be implemented using various solutions (e. g. high or low compensation of disturbances). The choice of solutions to these partial functions determines the solution to the overall function and its effect on the mechatronic system.

Let $PF_{leaf}$ be the set of all partial functions at the lowest level in the overall function hierarchy. Then, the selected solutions in the overall function at time $t_j$ (cf. Fig. 1.8, black circles) can be listed as a sequence of solutions to partial functions[35]:

$$s_{of}(t) = (s_{pf_1}, ..., s_{pf_k}), \tag{1.4}$$

with $s_{pf_i} \in S_{pf_i}$ and $1 \leq i \leq k = |PF_{leaf}|$.

The behavior of the overall system at time $t$ can be described by $V(t) = (\mathbf{x}_t, s_{of}(t), \mathbf{y}_t)$ with $\mathbf{x}_t$ as an input vector and $\mathbf{y}_t$ as an output vector. Consequently,

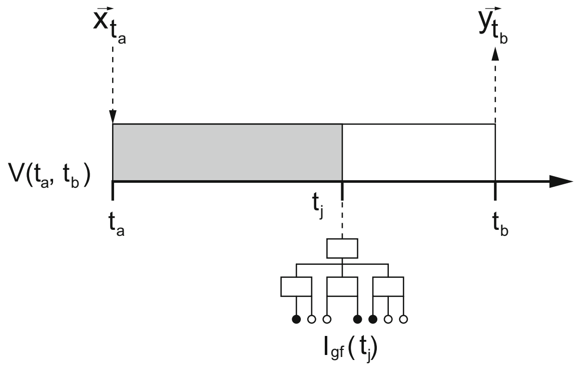


**Fig. 1.8** Planning sequence for the behavior of a self-optimizing system

the behavior of one time period is: $V(t_a,t_b) = (V(t_a),...,V(t_b))$ with $t_a < t_b$ (cf. Fig. 1.8).

The mechatronic system reacts to each input variable by converting it to the output variables; it does so by executing the currently implemented overall function (linked partial functions of the functional hierarchy, reactive behavior). The information processing takes time and there is a latency involved according to the response to input variables. Because of that, the effect of the overall function on the output variables is delayed. Since the system execution takes place under real-time conditions, this has to be taken into account during the development of self-optimizing systems.

For **autonomous control** [1] of the behavior, the system has to independently select solutions to the partial functions at certain times in order to achieve the desired effect (active behavior). The selection is made by considering specific objectives. This is the decision problem of a self-optimizing system: selecting solutions to partial functions which achieve these objectives with a high reliability result (goal-directed behavior).

### 1.4.2.1 Deterministic Planning

Mechatronic systems are able to execute a function in different ways. From the planning perspective, these different ways are distinguished from each other by how well they achieve the possible objectives and how they change the system state. We refer to these different implementations of functions as operation modes. In a mechatronic planning domain most relevant variables are numerical, hence we restrict the state vector to real valued numerical variables. A deterministic planning model for behavior-oriented self-optimization can be formulated as follows [36]:

- *OM* a finite set of available operation modes
- $S$ a finite set of possible system states
- $\mathbf{s} \in S$ a state vector with $s(i) \in \mathbb{R}$ for the i-th component

Furthermore for each operation mode $om \in OM$ exists:

- $prec^{om} := \{(x_{lower} < s(i) < x_{upper}) | x_{lower}, x_{upper} \in \mathbb{R}\}$ a set of preconditions which have to be true for executing *om*
- $post^{om}$ a set of conditional numeric functions to define the effects on the state variables of the subsequent state $s'$

A specific planning problem is finding a sequence of operation modes which describes a transition from an initial system state $\mathbf{s}_i \in S$ to a predetermined goal state $\mathbf{s}_g \in S$. So a single task of a mechatronic system is given as a 2-tupel $O = (\mathbf{s}_i, \mathbf{s}_g)$. A solution of the planning problem can be determined by applying a state space search algorithm (cf. [27]), for example [36].

### 1.4.2.2 Probabilistic Planning

Because of the uncertainty of environmental influences, probabilistic planning models are formulated based on the deterministic planning models. A probabilistic model for behavior planing for self-optimzing mechatronic system consists of probabilistic states $s^p$ with

$$range(s^p(i)) \rightarrow W(\mathbb{R})$$

for the value range (e.g. $0 \leq SOC_k \leq 100$ with $SOC_k$ for the state of charge of the mechatronic system in state $k$) and

$$distribution(s^p(i))$$

for the probabilistic distribution for state variable $s^p(i)$ (e. g. $\mathbb{P}(SOC_k \leq 50) = 0.25 \wedge \mathbb{P}(SOC_k > 50) = 0.75$). Furthermore, there are probabilistic variants of the operation modes *om* of the mechatronischen System with

- $in_s^{om} \subseteq pre^{om}$ for a subset of input variables and
- $out_s^{om} \subseteq post^{om}$ for a subset of output variables

Then for each output variable $o \in out_s^{om}$ a bayesian network (cf. [6]) $bn_o^{om}$ is created to formulate the probabilistic effect $\Delta$ on the state variable of the subsequent state $k+1$ by conditional probabilities (e. g. $\mathbb{P}(SOC_{k+1}^{\Delta} = +10 | windKmh \leq 20 \wedge SOC_k > 50) = 0.015$). With the definition of a lower respectively upper bound for critical state values (e. g. $SOC_{k+1} \leq 10$) branching points with likely violation of these bounds can be found and alternative plans can be genrated by using just-in-case-planning [35, 36, 38].

### 1.4.2.3 Hybrid Planning

The planning methods described above consider discrete system states and transitions relying, for instance, on average values or approximations. However, the continuous behavior can not be neglected for mechatronic systems. Therefore the necessity arises to integrate the continuous domain also in the planning process. Furthermore, planning for mechatronic systems has to cope with changing environmental conditions and imprecisions of a priori defined models during system operation which grow further with a widening planning horizon. For these reason continuous planning was combined with the discrete planning techniques presented above; the so called hybrid planning.

The hybrid planner uses the **discrete planning** techniques to generate an offline plan before the system starts its operation. For the RailCab this plan would, for instance, determine the course (sequence of track sections) to reach its destination and the parameter settings, i.e. the selected Pareto points based on the results of the multiobjective optimization. During system operation while executing the plan, however, deviations between the actual system state and those assumed by the offline plan can not necessarily be avoided due to the time that elapsed before the system reaches a certain state of the plan during execution. Also unforeseen conditions or changes of the environment may cause such deviations.

To counteract these shortcomings the hybrid planner simulates the system's behavior including the **continuous aspects** in an online manner. This simulation anticipates the future behavior of the system for a restricted time and allows to directly adapt the current action according to the simulation results. When the remaining part of the (discrete) offline plan is affected, the results of the just-in-case-planning may be used. If this is not possible an online replanning is initiated [4, 5, 18].

### *1.4.3 Self-optimization by Reconfiguration*

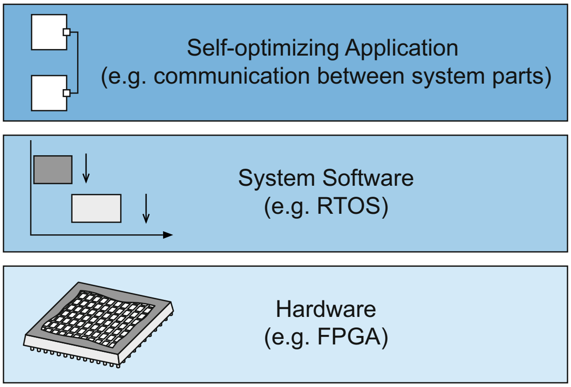
Stefan Groesbrink, Sebastian Korf, Mario Porrmann, Claudia Priesterjahn,
and Katharina Stahl

A self-optimizing system applies reconfiguration methods to adjust to changing requirements. In contrast to simple parameter changes, reconfiguration modifies the internal structure of a hardware or software system. When principles of self-optimization refer to the topology and structure of mechatronic systems, a reconfigurability of the system architecture or of dedicated system components is required. Reconfiguration decisions must be made autonomously. Therefore, parts of the classical design process have to be performed by the system at runtime: various implementation alternatives are available, from which the system selects the most appropriate realization (hardware or software) and the corresponding parameters.

Reconfiguration is executed on every system level: Self-optimizing Application, System Software, and Hardware. The levels of a self-optimizing mechatronic system are shown in Figure 1.9 and described in the following.

The **self-optimizing application** may be the execution of self-optimizing algorithms, e.g. finding optimal strategies to perform a task, or the communication between system parts. On the application level, reconfiguration means the exchange of software parts, e.g., switching between different software implementations to change the system behavior. Thereby, the self-optimizing application may change the requirements on the system software and its services. The **system software** interconnects the self-optimizing application and hardware and is composed of a virtualization layer and operating system. The virtualization layer is optional and enables the hosting of multiple operating systems on a single hardware platform. The system software supports the applications by reacting in a self-optimizing manner to

**Fig. 1.9** Levels of a self-optimizing system


Self-optimizing Application
(e.g. communication between system parts)
System Software
(e.g. RTOS)
Hardware
(e.g. FPGA)

the changing operating conditions of both the applications and the self-optimizing hardware. Self-optimization is introduced on the **hardware** level by means of dynamically reconfigurable hardware. Here, hardware reconfiguration means changing the functionality or the interconnect of hardware modules in microelectronic systems before and even during operation. Self-optimization in hardware must be encapsulated by the system software so that applications will execute without any perceivable interference. The system software must guarantee service supply in accordance to the given real-time constraints to enable hardware reconfiguration at runtime.

### Self-optimizing Application

The self-optimizing application is implemented in the Cognitive and Reflective Operator of the OCM (cf. Sect. 1.3.2). The Cognitive Operator gathers information about the system and the environment. It applies methods like learning and model-based self-optimization to optimize the system behavior. The Reflective Operator is the interface between the Cognitive Operator and the Controller. The interaction with the Controller requires operation in hard-real time and includes safety-critical tasks. This, in turn, demands a **safe software** that is free from design faults.

We therefore apply model-based software development to guarantee that the software satisfies all safety and real-time requirements. This means, the software is designed using models, the models are verified, and program code is generated, which preserves the verified properties.

On the level of the self-optimizing application, reconfiguration means the creation or removal of software components at runtime. This reconfiguration is specified by graph transformation rules (cf. Sect. 5.2.3.1) at design time. This allows to prove that no unplanned, e.g. unsafe, configurations are created at runtime.

The behavior of the components which execute the reconfiguration rules is modeled using state-based real-time behavior models. The reconfiguration rules are executed as side effects of this behavior. Therefore, the reconfiguration rules must not only guarantee to create no unplanned configurations but they must also satisfy the time constraints of the real-time behavior. To ensure this, we extended graph transformation rules by timing information and developed a verification approach (cf. Sect. 5.2.3.2) that takes into account the state-based real-time behavior, the reconfigurations, and the execution times of the reconfigurations.

System Software

The software models also allow computing application parameters such as the worst-case execution times [30]. In the context of mechatronic systems, the operating system has to manage the execution of the applications considering timeliness and predictability of the system behavior. It needs this parameter for the required real-time scheduling. Since efficiency is an important factor for operating systems for restricted environments such as mechatronic systems, operating system designers aim to deliver an application- or a domain-specific operating system with the objective to integrate required functionality only. ORCOS (**Organic Reconfigurable Operating System**) [19] is an example for an fully customizable real-time operating system at design time. Beyond design time configurability, the entire information processing process of monitoring, analyzing and reacting must be integrated into the self-optimizing operating system. In Sect. 5.5, we will present an extension of the ORCOS architecture that builds up the basis for online reconfiguration. However, self-optimization in the operating system is not restricted to react on requirement changes only. The operating system may also implement methods that can be applied to self-optimize the performance of the operating system (e.g. resource allocation strategy) as well as the overall system performance (e.g. resource utilization).

The operating system manages the use of hardware resources. This includes the abstraction of the underlying hardware which is usually done by implementing drivers. Dedicated interfaces specify the access to the hardware. In our approach, dynamic reconfiguration is provided by a combination of dynamically reconfigurable hardware and a reconfigurable real-time operating system (RTOS). The proposed hardware platform offers the fundamental mechanisms that are required to execute arbitrary software and to adapt the system to new requirements (e.g. by dynamic reconfiguration). The operating system triggers hardware reconfiguration as a reaction to varying requirements and decides whether a task is executed in software, in hardware, or in a combination of both.

Hardware

To adapt to changing environments, dynamically reconfigurable hardware is a key technology. Dynamically reconfigurable hardware can be classified as fine-grained or coarse-grained. Fine-grained reconfigurable architectures are typically based on **Field Programmable Gate Arrays** (FPGAs), which facilitates the System on Programmable Chip (SoPC) designs with a complexity of several million logic gates, several hundred kBytes of internal SRAM memory, and embedded processor cores. For the group of coarse-grained architectures we introduce reconfigurable embedded processors, which can change their internal structure to adapt to the currently needed environment.

The idea of dynamic and partial hardware reconfiguration is to reconfigure the hardware in a way that it maximizes the use of all available resources for the quired controller implementation. The information processing system is shared among all tasks, and offers limited resources with respect to memory, computational power, and energy. Any task may be composed of various sub-tasks and different realizations of these sub-tasks (e.g. different software implementations running on an embedded CPU and various hardware implementations for an integrated FPGA). Since all of these realizations have different computational requirements and different application characteristics, a control algorithm in a self-optimizing system can be understood as an optimal solution for the current internal and external objectives of the system. Therefore, in each new environmental condition of the mechatronic system, there is a controller architecture and a corresponding implementation variant that represent an optimal solution in this situation [45].

## 1.5 Structure of This Book

Mareen Vaßholz

In this Chapter self-optimizing systems were described briefly. It serves to show the potential of self-optimization for technical systems. In Chap. 2 examples of self-optimizing systems are presented that were developed in the Collaborative Research Center 614. They show the benefits that are provided by using self-optimization, but demonstrate its complexity as well. The resulting challenges for the development of these systems show the need for a design methodology presented in the following chapters. The different development tasks that have to be performed, are presented as a reference process for the development of self-optimizing systems in Chap. 3. This reference process is divided into the domain-spanning conceptual design and the domain-specific design and development phase. Chapter 4 depicts the domain-spanning development methods and tools. The ones relevant to the domain-specific design and development are presented in Chap. 5. The applications serve as examples for the description of the methods and tools for the development of self-optimizing systems. Chapter 6 gives a summary and an outlook over future work in the field of self-optimization and intelligent technical systems.

This book is one result of the research of the Collaborative Research Center 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" and is complemented by the book "Dependability of Self-Optimizing Mechatronic Systems". It focuses on tools and methods to ensure the dependability of self-optimizing systems during development and run-time. Throughout this book you will find cross-references, like [24, D.o.S.O.M.S. Chap. 2] , for detailed information on dependability specific methods and tools.

## References

1. DIN 19 226 Teil 1: Leittechnik - Regelungstechnik und Steuerungstechnik - Allgemeine Grundbegriffe. Deutsche Norm (1994)
2. VDI 2206 - Entwicklungsmethodik für mechatronische Systeme. Beuth Verlag, Berlin (2004)
3. Adelt, P., Donoth, J., Gausemeier, J., Geisler, J., Henkler, S., Kahl, S., Klöpper, B., Krupp, A., Münch, E., Oberthür, S., Paiz, C., Porrmann, M., Radkowski, R., Romaus, C., Schmidt, A., Schulz, B., Vöcking, H., Witkowski, U., Witting, K., Znamenshchykov, O.: Selbstoptimierende Systeme des Maschinenbaus. In: Heinz Nixdorf Institut, Universität Paderborn, vol. 234. HNI-Verlagsschriftenreihe, Paderborn (2009)
4. Adelt, P., Esau, N., Hölscher, C., Kleinjohann, B., Kleinjohann, L., Krüger, M., Zimmer, D.: Hybrid Planning for Self-Optimization in Railbound Mechatronic Systems. In: Naik, G. (ed.) Intelligent Mechatronics, pp. 169–194. InTech Open Access Publisher, New York (2011)
5. Adelt, P., Esau, N., Schmidt, A.: Hybrid Planning for an Air Gap Adjustment System Using Fuzzy Models. Journal of Robotics and Mechatronics 21(5), 647–655 (2009)
6. Ben-Gal, I.: Bayesian Networks. In: Encyclopedia of Statistics in Quality and Reliability (2007)
7. Binder, T., Blank, L., Bock, H., Bulirsch, R., Dahmen, W., Diehl, M., Kronseder, T., Marquardt, W., Schlöder, J., von Stryk, O.: Introduction to Model-based Optimization of Chemical Processes on Moving Horizons. In: Grötschel, M., Krumke, S., Rambau, J. (eds.) Online Optimization of Large Scale Systems - State of the Art, pp. 295–340. Springer, Heidelberg (2001)
8. Böcker, J., Schulz, B., Knoke, T., Fröhleke, N.: Self-Optimization as a Framework for Advanced Control Systems. In: Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics, Paris (2006)
9. Coello Coello, C.A., Lamont, G., Veldhuizen, D.V.: Evolutionary Algorithms for Solving Multi-Objective Optimization Problems, 2nd edn. Springer, Heidelberg (2007)
10. Comford, R.: Mecha.. what? IEEE Spectrum 31(8), 46–49 (1994)
11. Das, I., Dennis, J.: A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems. Structural Optimization 14(1), 63–69 (1997)
12. Dellnitz, M., Schütze, O., Hestermeyer, T.: Covering Pareto Sets by Multilevel Subdivision Techniques. Journal of Optimization Theory and Application 124(1), 113–136 (2005)
13. Dorociak, R., Gaukstern, T., Gausemeier, J., Iwanek, P., Vaßholz, M.: A Methodology for the Improvement of Dependability of Self-optimizing Systems. Production Engineering - Research and Developement 7(1), 53–67 (2013)
14. Dumitrescu, R.: Entwicklungssystematik zur Integration kognitiver Funktionen in fortgeschrittene mechatronische Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 286, Paderborn (2011)
15. Dumitrescu, R., Anacker, H., Gausemeier, J.: Design Framework for the Integration of Cognitive Functions into Intelligent Technical Systems. Production Engineering - Research and Developement 7(1), 111–121 (2013)
16. Dumitrescu, R., Gausemeier, J., Romaus, C.: Towards the Design of Cognitive Functions in Self-Optimizing Systems Exemplified by a Hybrid Energy Storage System. In: Proceedings of the 10th International Workshop on Research and Education in Mechatronics, Ostrava (2010)

17. Ehrgott, M.: Multicriteria Optimization, 2nd edn. Springer, Heidelberg (2005)
18. Esau, N., Krüger, M., Rasche, C., Beringer, S., Kleinjohann, L., Kleinjohann, B.: Hierarchical Hybrid Planning for a Self-Optimizing Active Suspension System. In: Proceedings of the 7th IEEE Conference in Industrial Electronics and Applications, Singapore (2012)
19. FG Rammig, University of Paderborn: ORCOS - Organic Reconfigurable Operating System, `https://orcos.cs.uni-paderborn.de/doxygen/html` (accessed August 12, 2013)
20. Frank, U., Gausemeier, J.: Self-Optimizing Concepts and Structures in Mechanical Engineering - Specifying the Principle-Solution (2005)
21. Frank, U., Giese, H., Klein, F., Oberschelp, O., Schmidt, A., Schulz, B.H.V., Witting, K.: Selbstoptimierende Systeme des Maschinenbaus. In: Heinz Nixdorf Institut, Universität Paderborn, vol. 155. HNI-Verlagschriftenreihe, Paderborn (2004)
22. Gausemeier, J.: From Mechatronics to Self-optimizing Concepts and Structures in Mechanical Engineering: New Approaches to Design Methodology. International Journal of Computer Integrated Manufacturing 18(7), 550–560 (2005)
23. Gausemeier, J., Frank, U., Donoth, J., Kahl, S.: Specification Technique for the Description of Self-optimizing Mechatronic Systems. Research in Engineering Design 20(4), 201–223 (2009)
24. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
25. Gausemeier, J., Steffen, D., Donoth, J., Kahl, S.: Conceptual Design of Modularized Advanced Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
26. Geisler, J., Witting, K., Trächtler, A., Dellnitz, M.: Multiobjective Optimization of Control Trajectories for the Guidance of a Rail-bound Vehicle. In: Proceedings of the 17th IFAC World Congress, Seoul (2008)
27. Ghallab, M., Nau, D., Traverso, P.: Automated Planning - Theory and Practice. Elsevier, Amsterdam (2004)
28. Gill, P.E., Jay, L.O., Leonard, M.W., Petzold, L.R., Sharma, V.: An SQP Method for the Optimal Control of Large-scale Dynamical Systems. Journal of Computational and Applied Mathematics 120, 197–213 (2000)
29. Harashima, F., Tomizuka, M., Fukuda, T.: Mechatronics - What is it?, Why and how? An Editorial. IEEE/ASME Transactions on Mechatronics 1(1) (1996)
30. Henkler, S., Oberthür, S., Giese, H., Seibel, A.: Model-driven Runtime Resource Predictions for Advanced Mechatronic Systems with Dynamic Data Structures. Computer Systems Science & Engineering 26(6) (2011)
31. Hestermeyer, T.: Strukturierte Entwicklung der Informationsverarbeitung für die aktive Federung eines Schienenfahrzeugs. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, Verlag Dr. Hut, München (2006)
32. Hestermeyer, T., Oberschelp, O., Giese, H.: Structured Information Processing for Self-Optimizing Mechatronic Systems. In: Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics, Setubal (2004)
33. Hillermeier, C.: Nonlinear Multiobjective Optimization - A Generalized Homotopy Approach. Birkhäuser (2001)
34. Isermann, R.: Mechatronische Systeme - Grundlagen. Springer, Heidelberg (2008)
35. Klöpper, B.: Ein Beitrag zur Verhaltensplanung für interagierende intelligente mechatronische Systeme in nicht-deterministischen Umgebungen. Ph.D. thesis, Fakultät für Wirtschaftswissenschaften, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 253, Paderborn (2009)

36. Klöpper, B., Aufenanger, M., Adelt, P.: Planning for Mechatronics Systems - Architecture, Methods and Case Study. Engineering Applications of Artificial Intelligence 25(1), 174–188 (2012)
37. Knowles, J., Corne, D., Deb, K.: Multiobjective Problem Solving from Nature: From Concepts to Applications. Springer, Heidelberg
38. Köpper, B., Sondermann-Wölke, C., Romaus, C.: Probabilistic Planning for Predictive Condition Monitoring and Adaptation within the Self-Optimizing Energy Management of an Autonomous Railway Vehicle. Journal for Robotics and Mechatronics 24, 5–15 (2012)
39. Landau, I., Lozano, R., M'Saad, M., Karimi, A.: Adaptive Control - Algorithms, Analysis and Applications. Springer, Heidelberg (2011)
40. Lückel, J., Hestermeyer, T., Liu-Henke, X.: Generalization of the Cascade Principle in View of a Structured Form of Mechatronic Systems. In: Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Como (2001)
41. Ober-Blöbaum, S., Junge, O., Marsden, J.E.: Discrete Mechanics and Optimal Control: An Analysis. Control, Optimisation and Calculus of Variations 17(2), 322–352 (2011)
42. Osmic, S., Trächtler, A.: Flatness-based Online Controller Reconfiguration. In: Proceedings of the 34nd Annual Conference of the IEEE Industrial Electronics Society, Orlando (2008)
43. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Kontruktionslehre. Grundlagen erfolgreicher Produktentwicklung - Methoden und Anwendung, 6th edn. Springer, Heidelberg (2005)
44. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Engineering Design - A Systematic Approach, 3rd edn. Springer, Heidelberg (2007)
45. Paiz, C., Hagemeyer, J., Pohl, C., Porrmann, M., Rückert, U., Schulz, B., Peters, W., Böcker, J.: FPGA-Based Realization of Self-Optimizing Drive-Controllers. In: Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, Porto (2009)
46. Schäffler, S., Schultz, R., Weinzierl, K.: A Stochastic Method for the Solution of Unconstrained Vector Optimization Problems. Journal of Optimization Theory and Applications 114(1), 209–222 (2002)
47. Schütze, O., Witting, K., Ober-Blöbaum, S., Dellnitz, M.: Set Oriented Methods for the Numerical Treatment of Multi-objective Optimization Problems. In: Tantar, E., Tantar, A.-A., Bouvry, P., Del Moral, P., Legrand, P., Coello Coello, C.A., Schütze, O. (eds.) EVOLVE- A bridge between Probability. SCI, vol. 447, pp. 185–218. Springer, Heidelberg (2013)
48. Strube, G.: Modelling Motivation and Action Control in Cognitive Systems. In: Mind Modelling, pp. 89–108. Pabst, Berlin (1998)
49. Witting, K., Schulz, B., Dellnitz, M., Böcker, J., Fröhleke, N.: A new Approach for Online Multiobjective Optimization of Mechatronic Systems. International Journal on Software Tools for Technology Transfer STTT 10(3), 223–231 (2008)

# Chapter 2
# Examples of Self-optimizing Systems

Joachim Böcker, Christian Heinzemann, Christian Hölscher, Jan Henning Keßler, Bernd Kleinjohann, Lisa Kleinjohann, Claudia Priesterjahn, Christoph Rasche, Peter Reinold, Christoph Romaus, Thomas Schierbaum, Tobias Schneider, Christoph Schulte, Bernd Schulz, Christoph Sondermann-Wölke, Karl Stephan Stille, Ansgar Trächtler, and Detmar Zimmer

**Abstract.** In this chapter, the benefits resulting from self-optimization will be described based on application examples from the Collaborative Research Center 614 “Self-optimizing Concepts and Structures in Mechanical Engineering”. First, the autonomous rail vehicle RailCab developed at the University of Paderborn is introduced. Then, the RailCab subsystems Self-Optimizing Operating Point Control, Intelligent Drive Module, Active Suspension Module, Active Guidance Module and Hybrid Energy Storage System and their test rigs are described in detail as well as an overall approach for Energy Management. The chapter concludes with the presentation of other development platforms such as the BeBot, an intelligent miniature robot acting optimally in groups, and the X-by-wire vehicle Chameleon with independent single-wheel chassis actuators. All the above mentioned demonstrators are used to validate the methods and procedures developed in the Collaborative Research Center. The experiences gained, provide direct input into further development and optimization of the design as well as the self-optimization process.

## 2.1 Rail Technology – RailCab

The **RailCab** system is an innovative self-optimizing transportation system for passengers and goods that allows comfortable travel on railways, while at the same time satisfying the desire for individual but energy-efficient mobility, with no distinction being made between local and long-distance traffic[2]. Important elements

[2] `http://nbp-www.upb.de`

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*, Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_2, © Springer-Verlag Berlin Heidelberg 2014

of the novel system are small, driverless vehicles, called RailCabs. These RailCabs can transport passengers or goods directly to their destination without the need for changing trains or reloading (cf. Fig. 2.1). The vehicles drive on demand and not according to a schedule. They can be ordered and configured via telecommunication services. In order to increase traffic flow on the lines, minimize energy consumption and maximize the flexibility, the RailCabs are not coupled mechanically, so that, on more frequented routes they can automatically form convoys, thus reducing air resistance, saving enormous amounts of energy, and increasing transportation capacity. The active steering of the vehicles along with passive track switches allows vehicles to move out of the convoy at track switches even in cases where the distance between the vehicles is small and at maximum speeds. As there are no stops or changes of trains, the RailCabs reach high average speeds and the transportation times are also short. The vehicles are designed in a modular way from standardized components that consist of intelligent function modules. The structuring of the RailCab into function modules can be conducted in the conceptual design (cf. Sect. 4.6). Therefore each vehicle can be scaled and flexibly adapted to its respective transportation tasks. The entire technical part is integrated into the undercarriage. This ensures that the system can comply with various specific transportation tasks. For the entire traffic system, a new and comprehensive logistics concept will be developed that employs a passive rail network with individually guided vehicles that make decisions autonomously. In order to provide a satisfactory informational base for these decisions, the RailCabs have positioning systems as well as mobile communication devices for communicating with one another and with the stationary facilities along the track. This gives the RailCabs the chance to detect congestions in advance and give them a wide forecast. Compared to conventional scheduling this will lead to a far better utilization of the system, because the RailCabs always travels along the route that is, among other objectives, optimal in terms of travel time. The economic efficiency of this system can be proven during the conceptual design based on the principle solution of the RailCab (cf. Sect. 4.8).

In order to test this entirely new RailCab system under realistic conditions a **test track** on a reduced scale of 1:2.5 was constructed at the University of Paderborn in 2003. It is 530 m long, consists of sections with an ascending slope of 5.3 % and a passive track switch, thus allowing in-depth checks and comprehensive trials of strategies for forming and disbanding convoys.

To date, two test vehicles have been built (see Fig. 2.2). Each of them is 3.4 m long and 1.2 m high and wide and weights 1.2 t. On the test track, vehicles can be operated at a maximum speed of 10 m/s. The test vehicles already comprise all the essential function modules of the future series-produced RailCabs. Every prototypical RailCab has two Active Guidance Modules, one for each single axle, and individual wheel sets, an active suspension system, two secondary elements of the doubly-fed linear-motor drive and an energy storage. For test operation, they are also equipped with a multitude of sensors and data-processing devices.

The operation of the prototypical RailCabs gives impressive proof of the technical feasibility of the whole concept and the self-optimizing behavior of the function modules as well as of the entire system. Besides, several Hardware-in-the-Loop

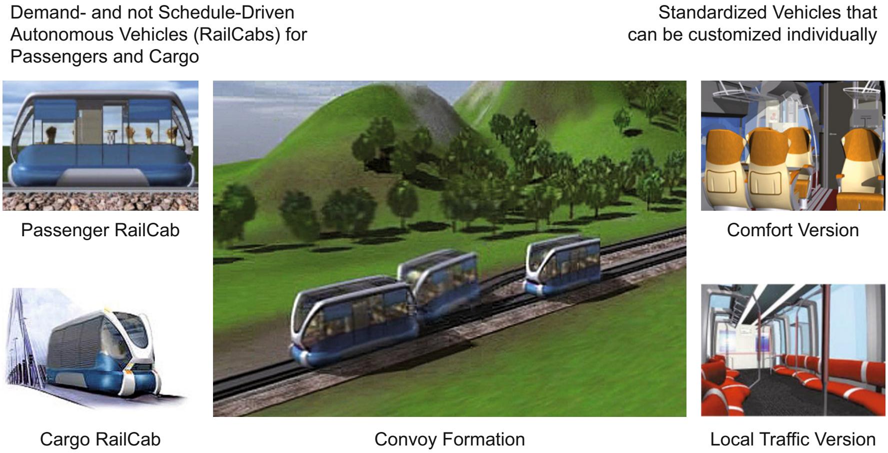
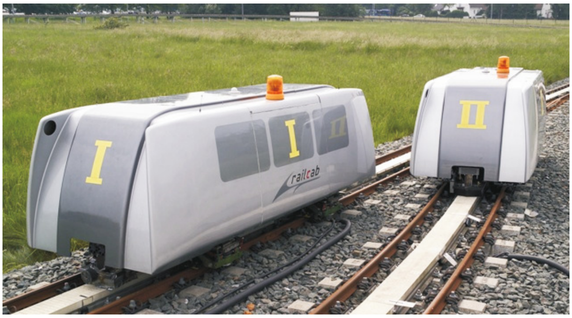

Demand- and not Schedule-Driven Autonomous Vehicles (RailCabs) for Passengers and Cargo
Standardized Vehicles that can be customized individually
Passenger RailCab
Comfort Version
Cargo RailCab
Convoy Formation
Local Traffic Version

**Fig. 2.1** The possible varieties of the RailCab

**Fig. 2.2** RailCabs on the test track (Scale 1:2.5)



(HiL) test rigs emulate the behavior of the function modules in order to explore innovative self-optimizing control strategies irrespective of the entire system. The following sections give a detailed description of the self-optimization approaches and the test rig implementations.

The development process of the RailCab and the different function modules served as input for the developed design methodology for self-optimizing systems presented in Chap. 3.

### *2.1.1 Self-optimizing Operating Point Control*

Christoph Schulte, Tobias Schneider, Bernd Schulz, and Joachim Böcker

The linear drive is one of the central components of the RailCab system. It consists of a doubly-fed linear asynchronous drive with an active track and offers both propulsion with a defined thrust independent of the wheel-rail-contact and contact-free energy supply of the vehicle [35].

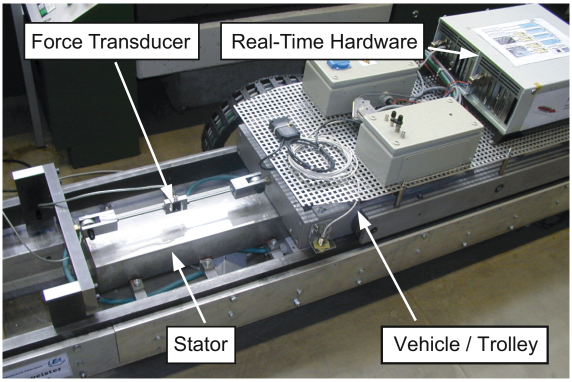


**Fig. 2.3** Test rig of the linear doubly fed asynchronous drive

Fig. 2.3 shows the 1:2.5 scaled test rig of the mentioned **linear doubly-fed asynchronous drive** which is used for research on the self-optimizing operating point control of the drive module. The behavior of this system is analyzed in detail with the illustrated construction of the doubly-fed linear drive shown in Fig. 2.3. It consists of an 8 m long straight test track with a varying airgap contour and an approximately 1 m long vehicle which can reach speeds $v_{\mathrm{m}}$ up to 2.5 m/s. The windings of the vehicle and track motor part are each fed by 3-phase power converters to set the desired thrust. For the control, a real-time processing unit is installed on the vehicle. The self-optimizing operating point control (SOAPS) composes working points by the inputs of speed control or from Energy Management of the vehicle. The determined working points are forwarded to the implemented control of the vehicle. The test rig also allows to measure the power consumption of the vehicle and the track synchronously as well as the thrust at standstill. Thus, it is possible to test the developed structures for self-optimization as well as the practical application of new optimization strategies and algorithms. In the following paragraph, an optimization problem is presented, which demonstrates some aspects of the self-optimization process.

According to Fig. 2.4, the overall power flow can be distributed in the individual stator and rotor side parts of the drive. Here, $p_1$ and $p_2$ are the stator and rotor side power, $p_{\mathrm{M}}$ is the mechanical power, $p_{\mathrm{B}}$ the power transferred to the vehicle, $p_{1,\mathrm{losses}}$ and $p_{2,\mathrm{losses}}$ are the dissipated power in the stator and rotor due to copper and iron losses. The propulsion force of the vehicle is defined by $F_{\mathrm{m}}$.

The ability to transfer energy with the doubly-fed induction drive is one of the great advantages for such an autonomous railway vehicle. But the energy transfer is also coupled with losses which increase the motor temperature and decrease the efficiency. Therefore the losses have to be determined precisely and hence an accurate loss model of the motor is required to ensure operation at an efficient working point.

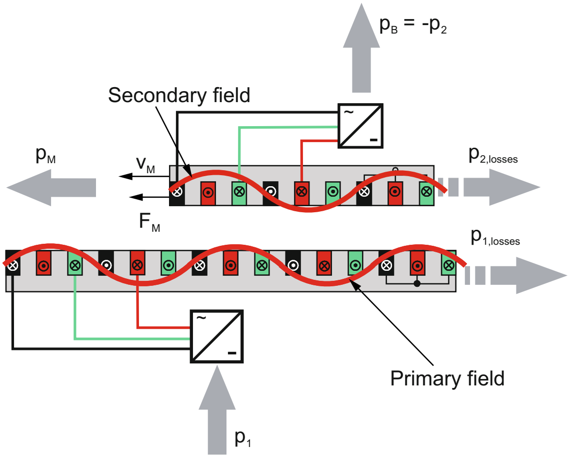


**Fig. 2.4** Power flow of the drive with corresponding losses

### 2.1.1.1 Determination of Optimal Operating Points

For the considered autonomous railway vehicle, a well defined thrust has to be developed. In this context, the thrust has to be understood as a constraint. Beside this constraint, there are also two conflicting objectives: On the one hand, a maximum power transfer is intended, which can be achieved by increasing the currents and/or the frequency. On the other hand, it is desirable to minimize the losses in order to increase efficiency. This requires low frequencies and currents.

During drive operation, a working point considering these two competitive objectives has to be selected, based on the given demands and surroundings. This dilemma is a classical example for a multiobjective optimization problem (MOP). In general, the objective functions will not have a common global optimum. Instead, the set of optimal trade-offs, the so-called Pareto set [35] is obtained.

Recently, combined set-oriented and path-following methods have been developed to treat time-dependent MOPs. Thus, the point determined by the decision heuristic is only valid for a fixed point in time. As it would be too time-consuming to approximate the entire Pareto set for many discretized points of time, we use numerical path following methods. These methods allow to solve parameter-depending equation systems. Thus, we can define a time-dependent path of Pareto optimal points by requiring that every point on the path has the same relevance of the objectives. To compute this path, the algorithm performs the so-called predictor-corrector steps [42] based on equation systems which are repeated until a given final time is reached. This allows the computation of entire Pareto sets for fixed points in time and the tracking of some chosen Pareto optimal points over time. These algorithms are very efficient and can even be used online for this application. Their mathematical background will be described in Sect. 5.3.1 and Sect. 5.3.4.

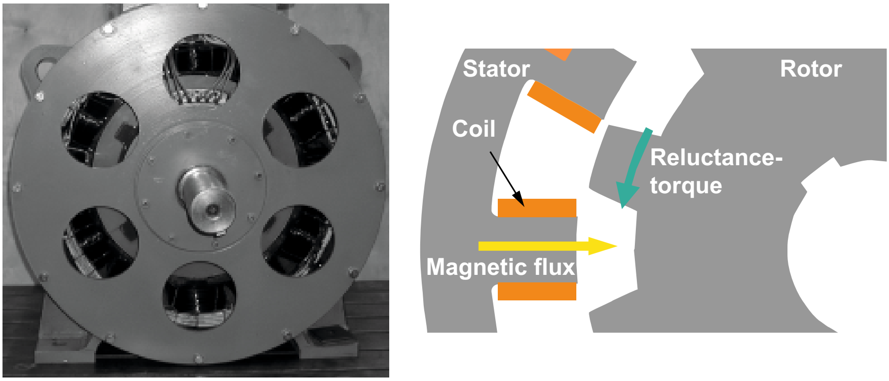


**Fig. 2.5** Photo and cross section of the switched reluctance motor

In the operating point assignment of the linear drive, a convex objective function (total losses) and a concave objective function (transferred power) emerge. The total losses are minimized, while the transferred power is maximized. The optimization variables are the stator and rotor side current and frequencies, time occurs as an additional parameter. In our case, the choice of one Pareto optimal point from the set depends on two parameters: the state of charge of the energy storage and the motor temperature. A low state of charge requires an increase of power transfer to charge the battery. This also increases the losses and the motor temperature respectively. In order to protect the drive against the risk of damage, it is advisable to take the motor temperature into consideration.

With this self-optimization working-point control it is now possible to take time-variant objectives into account and to adapt new objective functions during operation. Further, in the case of failure the selected working-point will be chosen sub-optimally but highly reliable, so that a further operation of the drive is enabled.

### 2.1.1.2 Alternative Drive System

In addition to the linear asynchronous drive an additional rotary switched reluctance motor test-rig was built [34], which can be seen in Fig. 2.5. In comparison to the asynchronous drive this type of motor offers a higher efficiency especially on passive tracks without stator windings. Here, an alternative working-point optimization was examined. The optimization goal in this case is represented by the maximization of the efficiency/minimization of the power loss, wherein the torque of the motor has to be kept constant [7]. To solve this optimal control problem a direct discretization method was used, which is based on the discretization of the method of Lagrange. This optimal control method, along with extensions e.g. to multiple objectives, is presented in Sect. 5.3.6 and Sect. 5.3.7.

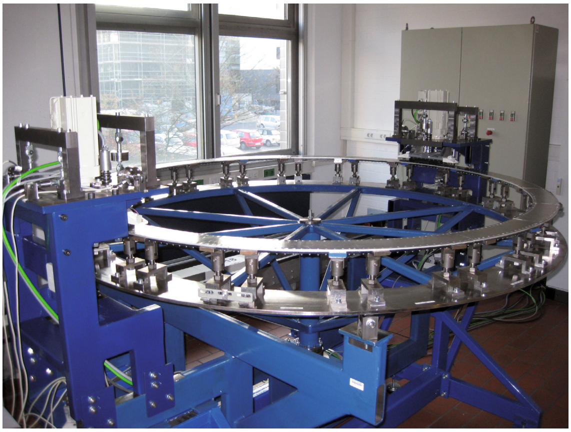


**Fig. 2.6** Test rig of the Intelligent Drive Module

## *2.1.2 intelligent Drive Module (iDM)*

Christian Hölscher and Detmar Zimmer

The drive of the RailCab is realized by a doubly-fed asynchronous linear drive. In addition to the force transmission, this direct drive enables **contactless energy transfer** to the RailCab [35]. The immovable part of the above named drive is the secondary part, which is installed on the track bed. The primary part is mounted at the bottom of the RailCab. Hence it is the movable part of the drive.

There is an **air gap** between the primary and the secondary part, which has a large influence on the electrical losses. Different influences such as incorrectly laid tracks and setting processes lead to displacements of the secondary parts, which results in a fluctuating air gap. In order to avoid a collision between the primary and the secondary part, a relatively large air gap can be chosen. Unfortunately this results in higher electrical losses and hence reduces the efficiency of the linear drive. The relationship between efficiency and the air gap is reciprocal. This implies an improvement of the efficiency by minimizing the air gap. A small but fluctuating air gap encourages a dynamic air gap adjustment . The Intelligent Drive Module (iDM) performs the task of an efficiency improvement by a track dependent dynamic minimization of the air gap. The main objectives are low energy consumption and a high degree of safety for the adjustment procedure. In order to validate an improvement of the efficiency by the iDM a test rig has been developed (Fig. 2.6).

The iDM is partitioned into two actuator groups (Fig. 2.7). Both consist of a primary part of the linear drive and an adjustment actuator. In addition spring assemblies are mounted to the actuator groups. The characteristic curves of the spring assemblies are adjusted to the normal force, which has a large influence on the secondary part. Hence, the adaption to the normal force reduces the load of the adjustment actuators. In case of an actuator failure, the linear drive will be raised by the spring assembly to avoid a collision between primary and secondary parts. Load cells on each actuator group measure the normal, the propulsion and the shear force.

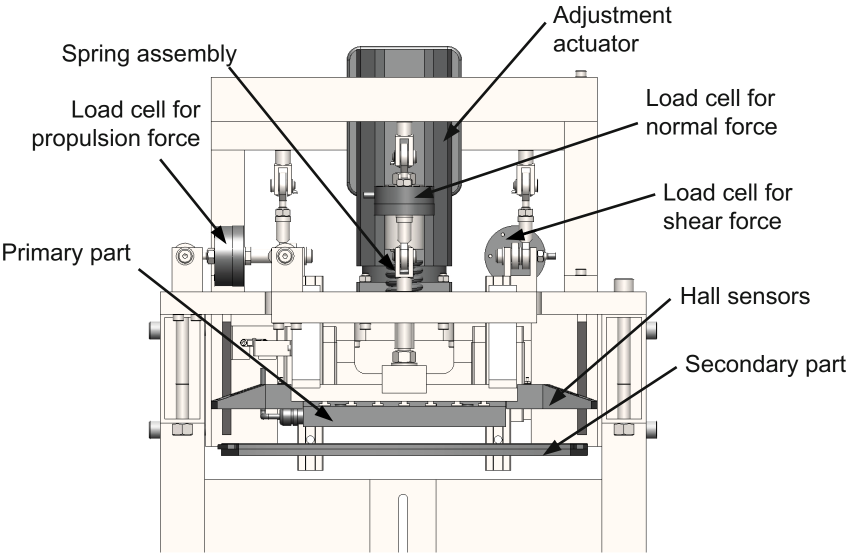

Adjustment actuator
Spring assembly
Load cell for propulsion force
Load cell for normal force
Load cell for shear force
Primary part
Hall sensors
Secondary part

**Fig. 2.7** Actuator group of the iDM test rig

They identify the characteristics of the forces depending on the air gap and the drive current. Hall sensors mounted on each of the primary parts measure the magnetic field of the secondary parts and determine the electrical rotation angle, the track velocity and the air gap. These measured data are essential for the control of the linear drives and the adjustment actuators.

In contrast to the RailCab, the primary part of the linear drive is mounted at the frame of the test rig and propels the secondary parts, which are mounted on the rotating track. That means, they are arranged in a circle and allow a continuous track simulation. The vertical position of the secondary parts can be adjusted manually to simulate different track characteristics.

The general objective of the iDM is to improve the efficiency of the linear drive by means of a dynamic air gap adjustment. The energy consumption of the iDM essentially depends on the environment, the operating point and the operating strategy. The influence of the environment takes effect through the track characteristics, the vertical profile and inherent influences like air resistance and friction. We consider three different track types. A rough track section comprises of many alternating displacements of the secondary parts. An obstacle track contains only a few displacements. A smooth track has no displacements. In this work we use a rough track with displacements of 3 mm. This is the maximum of secondary part displacements measured on the test track of the Neue Bahntechnik Paderborn. The vertical profile produces different load characteristics for the linear drive. The air resistance depends on the vehicle velocity. The operating point follows from the chosen velocity and needed propulsion force. The operating strategy optimizes the efficiency of the iDM. At first the two actuator groups are partitioned in a master and slave actuator group with different system behaviors. The master actuator group is the first group in the direction of travel. The air gap of the master linear drive has to be larger than the air gap of the slave drive to cover displacements which are not registered in the Track Section Control. The minimum allowed air gap of the master linear drive depends on the vehicle velocity and the maximum acceleration of the adjustment actuator and increases at a velocity threshold. The slave actuator group knows the actual track characteristics from the master actuator group. Due to the known track characteristics, the slave linear drive can adjust its air gap to the minimum value at any velocity. Furthermore the slave adjustment actuator uses a brake to reduce the actuator load. This brake produces a time delay and will only be used for the slave adjustment actuator. Two parameters, the master and slave adjustment actuators, will be determined for a track section. These two controller parameters $K_{p,up}$ and $K_{p,down}$ are further divided into parameters for the downwards and the upwards motion controllers. The optimal velocity $\omega_{track}$ for the track section is the third optimization parameter. These optimization parameters define the air gap of both the master and the slave actuator group. For the multiobjective optimization, two objectives are defined:

- The minimization of the absorbed energy of the adjustment actuators:

$$f_1 = E_{adj} = \int_0^T \sum_{i=1}^{2} P_{adj,i}(\tau)\, d\tau \tag{2.1}$$

- The minimization of the absorbed energy of the linear drives:

$$f_2 = E_{ld} = \int_0^T \sum_{i=1}^{2} P_{ld,i}(\tau)\, d\tau \tag{2.2}$$

The function $f_1$ describes the absorbed energy $E_{adj}$ of the adjustment actuators and $f_2$ the absorbed energy $E_{ld}$ of the linear drives.

In order to evaluate the objectives, a specific optimization model is used. This consists of a base model, an environment model and an evaluation model. The base model describes the behavior of the actuator groups.

The nonlinear characteristic of the normal and propulsion force is considered. This motor type is also assembled at the iDM test rig. A typical cascaded control is used for the actuators. The environment model describes track specific displacements of the secondary parts, the air resistance and friction.

The air resistance depends on the velocity and is adapted to test rig conditions. The evaluation model computes the objectives for each track section. A minimal air gap of 0.5 mm is ensured in the optimization process by means of a corresponding constraint term.

When raising velocity, the energy consumption $E_{ld}$ of the linear drives also raises, because of the increased demand for the propulsion force and the enlarged air gap. At a low velocity the energy consumption $E_{adj}$ of the adjustment actuators raises through an smaller air gap and the associated increased normal forces, which lead to an increased load of the adjustment actuators (2.8a). The calculated Pareto front

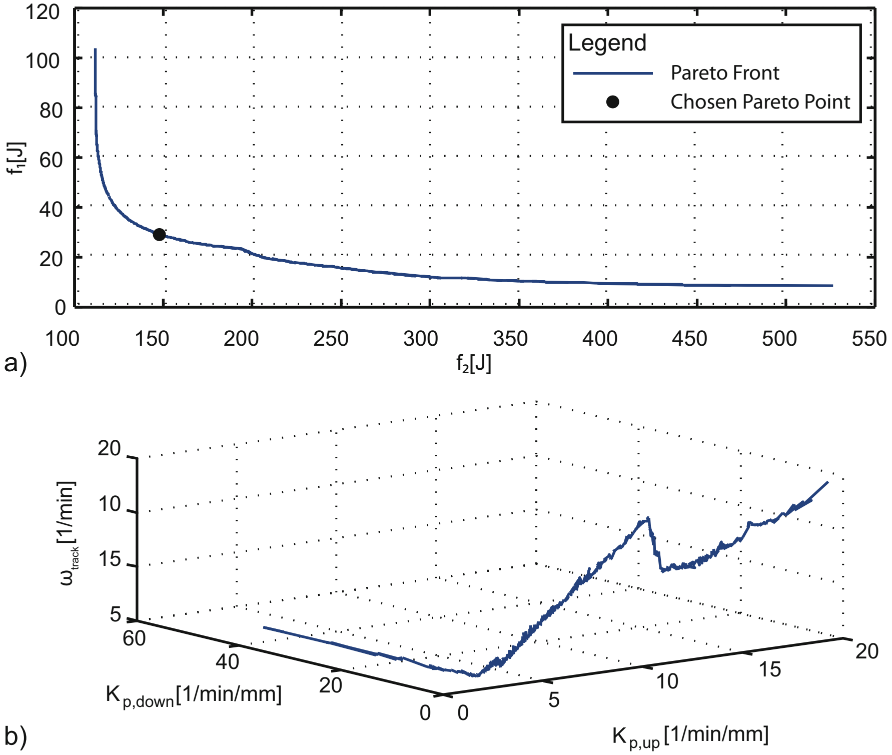


**Fig. 2.8** **a** Pareto front for the iDM, **b** Pareto set for the iDM

results from the set-oriented subdivision algorithm (cf. Sect. 1.4.1.1). The Pareto fronts of the different track sections can then be used as a basis for the hybrid planning approach described in Sect. 5.3.8.

Fig. 2.8b shows the Pareto set of the iDM for the sample track. These parameters are used in the superordinated optimization (cf. Sect. 2.1.6). Additional information of the iDM and the hierarchical optimization are described in [15] and in Sect. 5.3.2.

## *2.1.3 Active Guidance Module*

Christoph Sondermann-Wölke

As described in the introduction, RailCabs are able to form convoys and save energy using the slipstream of the vehicle ahead. Hence, an innovation is required to guide the RailCabs through a track switch if the RailCabs need to leave the convoy at track switches with an intended maximum velocity of about 180 km/h. Conventional switches are too slow to resolve this convoy, when certain RailCabs need to leave the convoy and continue their travel on a different track. Therefore, the passive switch was invented [41]. The **passive switch** offers the possibility for each RailCab to choose the direction independently by steering into the desired direction. The system module for this steering action is called the Active Guidance Module [5]. Each RailCab is equipped with two independently steering Active Guidance Modules. For the domain-spanning description of the system "Active Guidance Module" the specification technique CONSENS can be used. For example the application scenario "Steering into passive switches" can be specified (cf. Sect 4.1). In this application scenario, the description of the situation and the demanded behavior of the Active Guidance Module can be modeled. For example, the module should turn off the optimization in this scenario, to realize a controlled and safe drive through the passive switch. Further partial models of the principle solution like the active structure and the environment can be applied, too. Based on the models primary dependability analyses can be performed, like the early probabilistic reliability analysis based on the principle solution (cf. Sect. 4.7). For example, the engineer can detect that the actuator can be damaged, the flange could break or the RailCab could derail. Based on these identified causes and consequences, countermeasures can be determined to improve the dependability of the Active Guidance Module.

In addition to steering into passive switches, the guidance module actively controls the wheel guidance in normal tracks. By compensating disturbances like track irregularities and side wind flange contacts on straight tracks as well as in curved tracks are avoided, so that wear on wheels and rails is reduced. The clearance between the flange of the wheels and the rail heads is about 5 to 10 mm.

Therefore, a model-based optimization method for calculating trajectories for each Active Guidance Module is implemented [12]. The system of objectives can be defined by using the method for the design of the system of objectives (cf. Sect.4.4), which is based on the domain-spanning specification of the Active Guidance Module. Afterwards, set-oriented multiobjective optimization methods (for a detailed review cf. Sect. 5.3.1) have been applied to consider the objectives availability, comfort, energy efficiency and a limitation of the steering angle.

The planned trajectories are 10 m in length and consist of cubic splines parameterized by the decreasing number of knots over the whole trajectory. This concept was chosen, because only the first part of the trajectory is applied before it is replaced by replanning (receding horizon). The optimization is model-based because of the underlying model of the track system and, even more important, the model of the lateral dynamics of the RailCab vehicle. Depending on the current situation, a suitable trajectory is derived from a calculated Pareto set by weighting the objectives. This Pareto set is calculated offline considering a model of the RailCab, the track including track irregularities, and different velocities. Online, one Pareto point is chosen according to the desired objectives of the guidance module. This Pareto point leads to the calculation of the desired trajectory. A trajectory increasing availability will be close to the center line of the clearance (to avoid flange strikes), whereas a trajectory maximizing the energy efficiency will cut curves and increases the probability of flange strikes. The selection of Pareto points and their online adaptation to the actual system and environment state could also be supported by the hybrid planning approach described in Sect. 5.3.8.

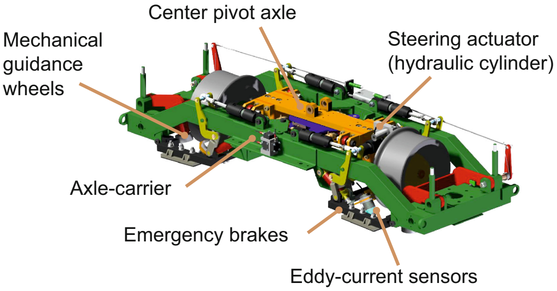


**Fig. 2.9** Main components of the Active Guidance Module

This trajectory is the set value for the underlying control strategy, which is executed in the controller of the Active Guidance Module. One module with its most important components is shown in Fig. 2.9. It consists of the axle carrier in which the center pivot axle is suspended. Each set of wheels uses loose cylindrical wheels to reduce slip and wear on wheels. The axle is actuated by a servo-hydraulic cylinder with an integrated displacement sensor.

For control, every guidance module features four eddy-current sensors, which measure the distance between the flange of the wheels and the rail heads. Two diagonally arranged sensors are coupled and the mean of both is calculated to get more precise information about the position of the axle relative to the track. This sensor data is used to calculate the current deviation to the planned trajectory and thus the required steering angle to reduce the deviation. The steering angle is set by the steering actuator, a hydraulic cylinder. For this purpose, a PI-controller is used. If the self-optimization process fails and no trajectory is calculated, the center line of the clearance is used. In advance, a feed-forward control is chosen, which calculates the desired steering angle based on the straight and curved track sections. If the longitudinal position, which is determined by incremental sensors and a proximity switch, fails, then none of the control strategies can be executed. To avoid uncontrolled behavior of the guidance modules in this case, the axles are fixed. This increases the wear on wheels and rails dramatically and the velocity of the RailCab should be reduced. Fixing axles does not lead to a standstill of the RailCab.

Further information regarding different control strategies and the behavior of the Active Guidance Module in case of sensor failures can be found in [38].

## *2.1.4 Active Suspension Module*

Jan Henning Keßler and Ansgar Trächtler

The RailCab is equipped with an Active Suspension Module that is designed to increase passenger comfort. The task of this module is to suppress the undesirable effects of frequent bumps and other agitations of the railway. For the development of innovative self-optimizing control structures, a Hardware-in-the-Loop (HiL) test rig was built which emulates the suspension of a half-vehicle (Fig. 2.10). The test rig consists of a body mass that represents the coach body with its three degrees of

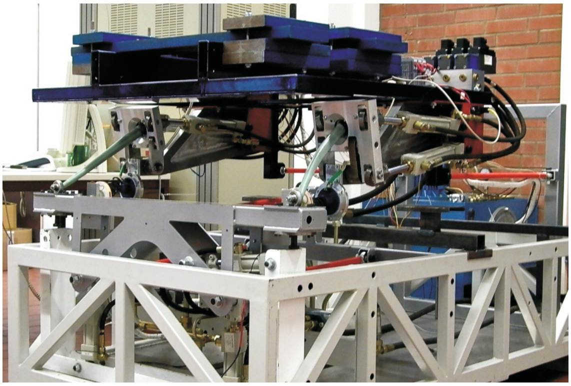


**Fig. 2.10** Test rig of the Active Suspension Module

freedom in vertical, horizontal and rotational (body roll) direction. Beneath the coach body there are two symmetrically mounted actuator groups, each with a sophisticated kinematics guide, a passive fiberglass reinforced polymer spring (GRP-spring) with a low damping characteristic and three hydraulic cylinders. By deflecting the spring base actively, it is possible to exert additional forces on the right and left side of the coach body in the vertical and horizontal direction in order to damp the coach body's motion in each degree of freedom. A Skyhook controller calculates these damping forces using the body velocities and the three controller parameters representing the damping characteristic of each degree of freedom of the coach body. The chassis framework that can be displaced by three hydraulic cylinders. This is used to simulate the railway excitations. The chassis framework is connected to the lower end of the GRP-springs.

#### 2.1.4.1 Problem Definition and Self-optimization Approach

The purpose of the self-optimizing active suspension system is to increase passenger comfort by minimizing the accelerations of the body, while reducing the energy consumption of the actuator modules. The lower the acceleration of the body, the higher the damping forces and the power demand of the actuator modules. This leads to an optimization problem with two conflicting objectives: "minimize body accelerations" and "minimize power demand". By varying the three Skyhook controller parameters, the system characteristics can be adapted to the two objectives. Thus for an optimization of the system the controller parameters are also the optimization parameters. The two objective functions are typically chosen as a mean value of characteristic signals, i.e., they are given by the integral functions

$$f_1 = \frac{1}{T}\int_0^T \sum_{j=1}^{6} P_{hyd,j}(\tau)\, d\tau \tag{2.3}$$

$$f_2 = \frac{1}{T}\int_0^T \sum_{i=1}^{3} |w_i(a_i(\tau))|\, d\tau \,. \tag{2.4}$$

The function $f_1$ describes the power demand. It comprises of the hydraulic power $P_{hyd}$ of the six cylinders. The function $f_2$ describes the level of comfort concerning the weighted body accelerations $a_i$ in the aforementioned three degrees of freedom. The weighting filters $w_i$ are explained in [1].

Multiobjective optimization has been proven to be an effective technique for computing optimal system configurations. The solution of the multiobjective optimization problem is given by a set of points, the so-called Pareto set. The image of the Pareto set are the optimal compromises of the objectives, called the Pareto front (cf. Sect. 1.4.1.1). The optimization is based on a complex nonlinear model, which emulates the suspension system in detail. It comprises the differential equations of the dynamical behavior of the active suspension system and a synthetic model of the railway disturbances that are assumed to be stochastic. Most of the real disturbances are unknown and differ from the synthetic disturbances of the optimization model. Thus the task of self-optimizing control is to choose and adjust points on the Pareto set, i.e. controller parameters, that guarantee an optimal behavior in terms of the desired objectives despite the effects of unknown disturbances over time.

The active suspension system also serves as an application example for several self-opimization methods. Two methods based on the hierarchical system structure have been applied. First, hierarchical modeling has been used to compute a simplified model of the suspension system by means of parametric model-order reduction, cf. Sect. 5.3.3. Second, hierarchical optimization has been applied to a coupled system that consists of the active suspension system and the Intelligent Drive Module, see Sect. 5.3.2 for more details. Furthermore, varying crosswind conditions which affect the Active Suspension Module can be formulated as a parametric multiobjective optimization problem to compute so-called robust Pareto points (cf. Sect. 5.3.4 and Sect. 5.3.5). The hybrid planning approach (cf. Sect. 5.3.8) can then be used to determine a sequence of Pareto points and adapt them online to the actual environment conditions, which might differ from those assumed during the multiobjective optimization.

### 2.1.4.2 Objective-Based Controller

Concerning unknown and continuously varying disturbances of the railway, it is not viable to control both values of the conflicting objective functions at the same time, because the desired values are either unattainable or better solutions for both objectives exist. Finally the aim of the objective-based controller is to drive the system toward a desired relative weighting $\alpha$ of the objective values and to adjust deviations of this weighting quickly.

The structure of the objective-based control scheme is illustrated in Fig. 2.11. It can be seen as a hierarchical control structure. The classical Skyhook controller, which parameters are adjustable at runtime, forms the lower control loop. The upper loop is given by the objective-based controller. The current values of the objective functions $F(y(p^*))$, are computed using the output $y(p^*)$ of the dynamic system. These values are transformed via the function $s^{-1}$ to the domain of the $\alpha$ parameterization, yielding $\alpha_{cur}$. The variable $\alpha_{cur}$ is given by the relative length of the

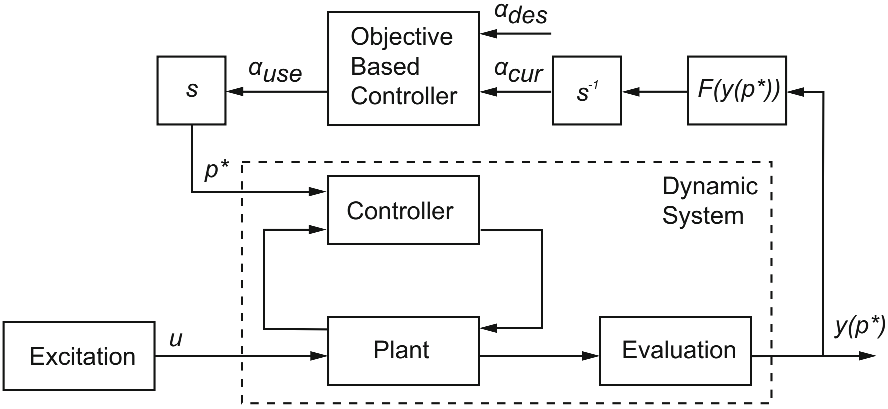


**Fig. 2.11** Structure of the objective-based controller

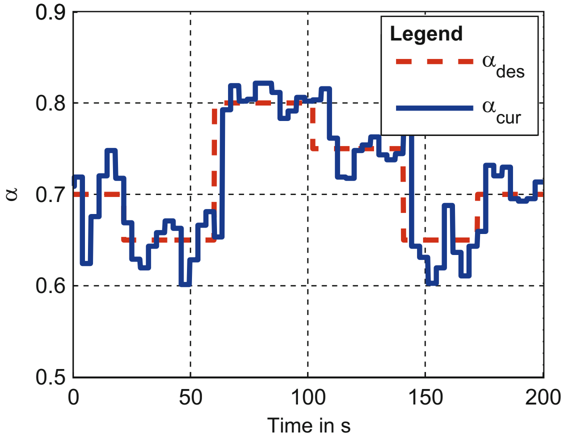
orthogonal projection of the current point in objective space compared to the length of a straight line between the terminal points of the Pareto front. The current and desired $\alpha$ values serve as inputs to the objective-based controller, which outputs the $\alpha_{use}$ value. The transformation $s$ is then applied to find the corresponding controller configuration $p^*$ from the Pareto set, which is used in the Skyhook controller.

The control structure uses two different sample rates. While the lower control loop operates with a very fast sampling time, the objective-based controller of the upper loop operates in discrete time $T$ with a significantly slower sampling rate. The sample time $T$ depends on the average amount of time necessary for the objective functions to converge to a relatively constant value in response to a change in the Skyhook controller parameters effected by the upper loop's previous cycle. The objective-based controller is realized as a discrete PI-controller. A detailed description of the controller design and the realization is presented in [24] and [23].

#### 2.1.4.3 Test Rig Implementation

The practical results at the Hardware-in-the-Loop test rig and the performance of the objective-based controller show that the sample time $T$ of each upper loop's cycle has to be 3.5 seconds so that the objective function values are converged. The chassis framework simulates disturbances of the railway that change over time in order to ensure unpredictable and non repeating characteristics. The results are illustrated in Fig. 2.12. The desired value $\alpha_{des}$ of the relative weighting of the two objective functions changes over time. The objective-based controller is able to adjust the system characteristics by calculating new configurations from the Pareto set in response to a change in $\alpha_{des}$. The response speed of the controller is approximately 1 to 2 sample times $T$. The controller is also able to maintain the current relative weighting $\alpha_{cur}$. The fluctuation of $\alpha_{cur}$ is the result of unpredictable and

**Fig. 2.12** Desired and current $\alpha$ values over time



continuously changing disturbances, but is sufficiently close to the desired value $\alpha_{des}$.

## 2.1.5 *Hybrid Energy Storage System (HES)*

Karl Stephan Stille, Christoph Romaus, and Joachim Böcker

The on-board power system of the RailCab is primarily fed by the power transfer via the doubly-fed linear motor. As the power transfer is not sufficient under all operating conditions, an on-board energy storage is necessary to continuously supply the vehicle.

The requirements of the energy storage are high energy storage to offer long operation as well as high power while charging and discharging to meet the demands. When installed in the vehicle, the mass and volume also have to be small. Low costs, a high efficiency and operating safety are implicit. As existing energy storage technologies do not satisfy these requirements, a combination of **batteries** and **double layer capacitors** (DLC) in a Hybrid Energy Storage System (HES) was chosen. The batteries serve as long term energy storage with high specific energy, while the DLCs provide high specific power for short term. The storages are each connected via bi-directional power converters to a common DC-link, which supplies the motor and other loads of the vehicle (Fig.2.13).

### 2.1.5.1 Energy Management

With two independent energy storages, a degree of freedom exists for the distribution of power. Energy Management is therefore necessary to benefit from this. It should react to varying influences from the environment of the vehicle and adequately adapt its behavior. Depending on the situation, different objectives can be important.

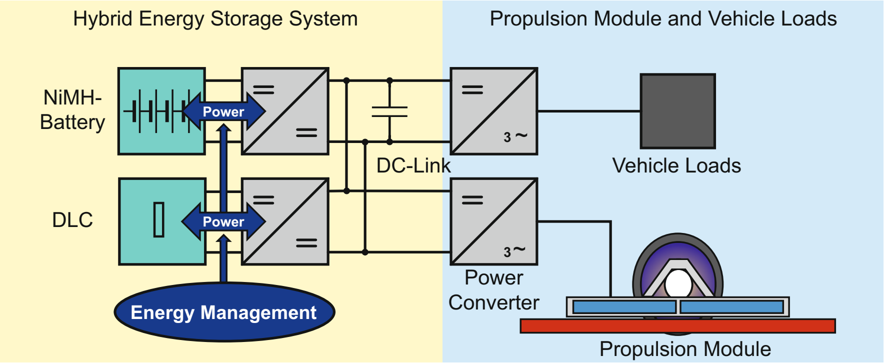


**Fig. 2.13** Structure of the Hybrid Energy Storage System

As the RailCab is automatically operated, the future power demand can be sufficiently calculated ahead by a predictor leading to the possibility of using optimal optimization methods, that offer very high quality results [31]. For applications with stochastically characterized power demands, suboptimal strategies can be used as well [22, 30, 32]. Statistical planning techniques (cf. Sect. 5.3.9) could be used in this context to learn power demands on certain track sections from previous observations.

The travel of the vehicle is divided into short sections of about 1-5 minutes depending on the frequency and significance of changes in the power profile provided by the predictor. The optimization of the operating strategy is then carried out for each section. It can be described as an optimal control problem, stating the distribution of the on-board power of both storage devices at discrete points in time, represented by the battery current as optimization variable. The time between these discrete points is calculated depending on the amount of energy transferred from or to the energy storage and the variation of the demanded power.

Two objectives are of particular importance:

- Minimization of the normalized energy losses $E_{\text{losses}}$ of the HES – objective function 1:

$$f_1 = E_{\text{losses}} = \int_{t_0}^{t_0+T} p_{\text{losses}} \mathrm{d}t \tag{2.5}$$

- Maximization of the power reserve to compensate for unexpected power peaks, thus increasing availability, if there are deviations from the predicted power profile. This is expressed by the minimization of an associated second objective function $P_{\text{res}}$, which represents the deviation of the DLC's state of charge $SOC_{\text{DLC}}$ from a medium value $SOC_{\text{med}}$:

$$f_2 = P_{\text{res}} = \frac{1}{T} \int_{t_0}^{t_0+T} \left( \frac{SOC_{\text{DLC}} - SOC_{\text{med}}}{SOC_{\text{DLC, max}} - SOC_{\text{med}}} \right)^2 \mathrm{d}t \tag{2.6}$$

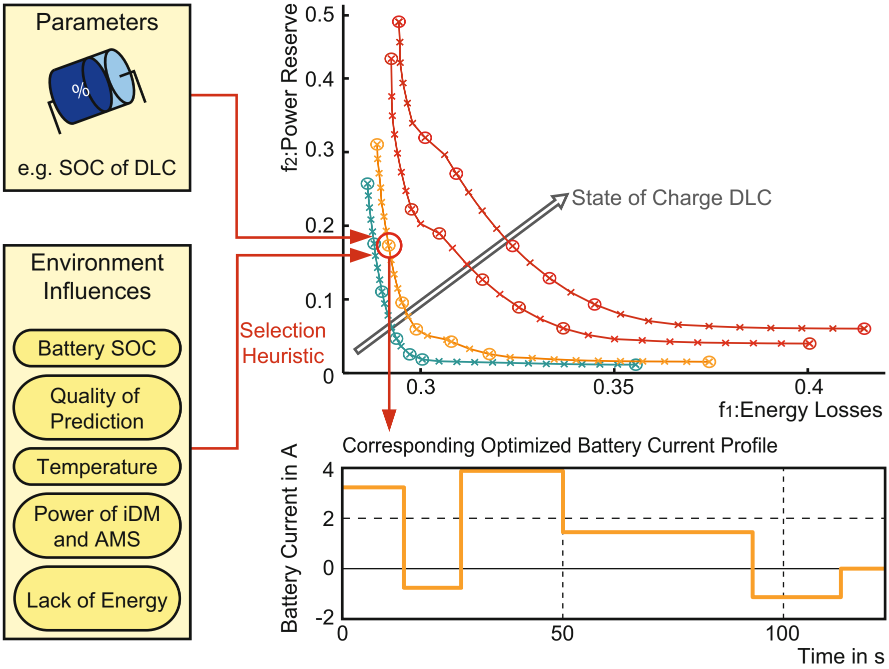


**Fig. 2.14** Structure of the multi objective optimization of HES

Mathematically speaking, this problem is called a multi-objective optimization problem (MOP), because the objectives compete with each other. There are different approaches to solve MOPs. In the case of the Hybrid Energy Storage System, a combination of the set-oriented multiobjective optimization methods described in Sect. 5.3.1 and a discrete heuristic search are applied. The set-oriented methods offer high quality results and an overview of the possible objective values by approximation of the complete Pareto set (Fig. 2.14). However, as they are computationally costly, they are only applied to well-known, often recurring track sections with specific power profiles precalculated in advance, e.g. in shuttle or suburban rail service. For other track sections, the heuristic search is applied online, showing inferior quality of results due to discretization of the optimization variable and only local optimality [31]. Currently a new on-line Pareto search, which is described in [40], is being implemented.

### 2.1.5.2 Self-optimization

As shown in Fig. 2.14, both objective functions can be varied in a wide range. For example, the losses $f_1$ can be reduced by 23 % inbetween the different solutions, $f_2$ by even 98 %. However, both objectives can not be minimized at the same time. So with two objectives and different possible solutions, the question arises which solution should be selected. For classic operating strategies, the weighting of objectives has to be fixed during controller design, accepting a trade-off for all possible situations. Adaptive strategies can ensure this weighting even under a varying system environment. But the weighting of objectives is fixed, which leads to suboptimal solutions in many cases.

With the concept of *self-optimization*, the relevance of objectives is selected online during the operation of the RailCab. Thus, the system can autonomously react to varying surroundings and choose the appropriate solution. This is demonstrated below by the example of a RailCab traveling twice on the same track section with the same velocity and thus the same power demand from the HES.[3] During the first travel, however, the power demand is unassured and can only be estimated with high uncertainty, e.g. as no data of previous travels is available or as there are distinct variations in the predicted power, e.g. due to weather conditions. Then, the same track section is passed over again, this time with a reliable prediction of power due to favourable conditions.

The self-optimization Energy Management now performs the *3 steps of self-optimization* (Fig. 2.14) for each following track section:

1. **Situation analysis:** Relevant parameters like costs for power transmission, reliability of prediction or state of charge of the storages are analyzed.
2. **Determination of relevance of objectives:** The relevance of objectives is determined e.g. by heuristic rules. Objective one ("minimize energy losses") is chosen preferably when energy is low on the on-board system, costs of power transmission are high and the power prediction is dependable. Otherwise, preference shifts to the second objective to offer a power reserve.
3. **Adaptation of system behavior:** The corresponding optimal operating strategy is determined. In case of well-known sections and pre-calculated strategies by the set-oriented methods, a solution is taken from a database, otherwise the heuristic search is applied, weighting the relevance of objectives according to the former step. In the second case the hybrid planning approach (cf. Sect. 5.3.8) can be used.

The result of the self-optimization process is presented in Fig. 2.15 and compared to a strategy with static System of Objectives for the described travels. On the first travel, the self-optimization strategy mainly follows objective $f_2$ to ensure a high power reserve to compensate for the uncertainties in power prediction, reaching better values up to 82 % cp. to the static strategy. Objective $f_1$ is disregarded. During the second travel, now objective $f_1$ is prioritized, improving by 23 % cp. to the strategy used so far. The mean power dissipation is lower by 13 % cp. to the static strategy. Obviously, the objectives relevant during the respective travel are better achieved by the self-optimization strategy compared to the operating strategy with static System of Objectives.

Additional details on the optimization of power flow in the Hybrid Energy Storage System are described in [30, 31, 32, 40].

[3] The same track section was chosen to clearly compare the results of self-optimization. In real operation, the mentioned conditions will apply to different sections as well.

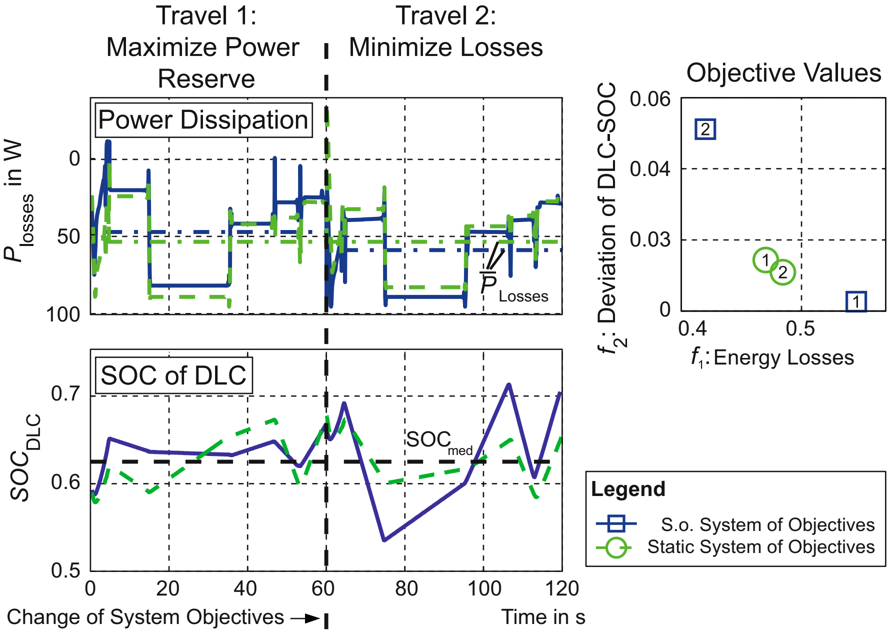


**Fig. 2.15** Self-optimization operating strategy compared to an operating strategy with static System of Objectives

Self-optimization adapts the relevance of competing objectives according to the environment of the system. It offers low losses in uncritical situations while ensuring a secure and dependable operation at demanding travels. Thus, the self-optimization strategy superiorly meets the objectives that are relevant to the respective situation compared to static strategies, achieving a distinctly enhanced quality of results.

### *2.1.6 Crosslinked Test Benches*

Karl Stephan Stille and Joachim Böcker

The RailCab's energy needs are transferred via contactless power transfer in the Intelligent Drive Module (cf. Sect. 2.1.2). As the energy transfer is not directly controllable, the Hybrid Energy Storage System (cf. Sect. 2.1.5) is necessary to supply the entire vehicle. The RailCab consists of several Mechatronic Function Modules explained in the sections before, each one using self-optimization for the local optimal operation.

For an efficient operation, the RailCab needs to find an overall optimal operation point. This is accomplished by a distributed crosslinked Energy Management.

As the test benches are situated in different locations, influences of the function models on each other are not directly verifiable. For this purpose all test benches have been interconnected via Ethernet.

Reflecting the prior sections, the mechatronic function modules of the RailCab have the following conflicting objectives:

- Intelligent Drive Module
    1. minimize power loss
    2. maximize transferred power
- Hybrid Energy Storage System
    1. minimize power loss
    2. maximize power reserve (ability to react on unforseen power demands)
- Active Suspension Module
    1. minimize energy consumption
    2. minimize chassis acceleration (which is the maximization of comfort)

The common objective of all modules is "minimize energy consumption". From this point it is possible to globally optimize the energy requirement of the RailCab on AMS level, as well as finding a Pareto trade-off between the efficiency needs and the other objectives.

#### 2.1.6.1 Structure of the Crosslinked Test Benches

At the beginning of the development it is necessary to create a common understanding of the system between the developers involved, to be able to realize the Crosslinked Test Benches. Therefore the system is modeled using the specification technique CONSENS (cf. Sect. 4.1). Figure 2.16 shows its active structure and the interaction of the function modules. The environmental influences are not shown to keep the structure simple. In the lower part the three test benches, the Intelligent Drive Module (cf. Sect. 2.1.2), the Active Suspension Module (cf. Sect. 2.1.4) and the Hybrid Energy Storage System (cf. Sect. 2.1.5) can be seen. Only those three test benches communicate with each other in hard real time as they are test benches with real mechanical or electrical hardware, exchanging information about system states and instantaneous power. The data transmission of the instantaneous power emulates the power flow in the vehicle's electrical system which can not be modeled by real power flow as the test benches are not located close to each other.

The optimization block above does not have a controller or a reflective operator. The complete optimization is done in soft real time, stating that the optimizer is a cognitive operator for the system on AMS level.

Only dependability functions are added to each part of the system to check for malfunctions and to enable real time capability reactions. This way they are implemented as reflective operators.

Data-handling of the whole vehicle is combined in the vehicle management. The track data processing communicates with data servers near the track fetching information about velocity limits, slopes, environmental data and track/stator positions measured previously by other vehicles on the same section. While fetching information about the next section, track data processing transmits its own measured data of

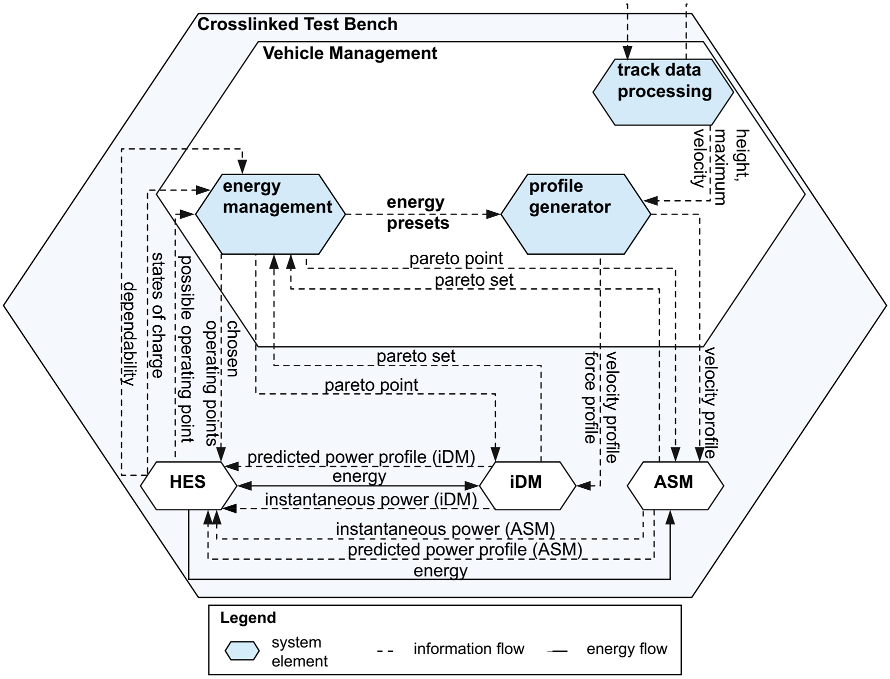


**Fig. 2.16** Active structure of the Crosslinked Test Benches

the previous section. By this process the database on the data server near the track is always up-to-date.

### 2.1.6.2 Optimization

The optimization itself consists of an Energy Management and a profile generator. The **profile generator** creates profiles of track, velocity, slope and needed thrust for the following section based on the information from the track data processing and energy presets given by the Energy Management. Those energy presets are evaluated based on the needed energy for the travel, the states of charge of the energy storage devices and the possibility to get energy transmitted from the stator.

In addition, to give presets to the profile generator, the **Energy Management** coordinates the overall multiobjective optimization of the whole system. For this, it receives Pareto sets from the MFM which were created based on the current information about the following track section.

As the HES needs to know the power demand of the other MFMs, its Pareto set can not be calculated nor chosen prior to get information about the power profile from the iDM and the ASM.

To break this optimization loop which is not guaranteed to converge, a hierarchical optimal control has been chosen. It combines reduced models of the test benches into one optimization problem. As components on one level of optimization are not

**Fig. 2.17** RailCab joining a convoy at a switch

allowed to depend on each other, the HES has been raised to an intermediate level between the mechanical test benches and the Energy Management System.

In order to exploit the hierarchical problem structure and to combine the results obtained by the multiobjective optimization, the hierarchical hybrid planning (cf. Sect. 5.3.8) can be used to select appropriate Pareto points determining the parameter settings for such a hierarchical mechatronic system.

### *2.1.7 Convoy Mode*

Christian Heinzemann and Claudia Priesterjahn

The objective of the convoy mode (cf. Sect. 2.1) is the optimization of the energy consumption of the RailCab [13]. As a result of the small distances between the vehicles, each RailCab drives in the slipstream of the RailCab which is driving directly in front. This reduces the air resistance and thereby reduces the energy needed for driving.

If a RailCab is driving alone but can benefit from convoy mode, it needs to adapt its behavior in order to enter a convoy. After the behavior adaptation, the RailCab may realize the behavior of the convoy *coordinator* or a convoy *member*: At any time, one RailCab needs to serve as a convoy coordinator. The **convoy coordinator** drives ahead and defines, e.g. the speed for the convoy. In addition, it needs to notify all other RailCabs in the convoy, called **convoy members**, about acceleration and braking maneuvers to ensure convoy stability and prevent collisions. If another RailCab wants to join the convoy, the coordinator needs to decide at which position the RailCab may safely enter the convoy depending on its braking characteristics (cf. [10, D.o.S.O.M.S. Sect. 3.2.10] , [6]).

The behavior of the convoy coordinator and convoy members is realized by software and communication protocols to coordinate the RailCabs within the convoy. This software realization is necessary, because RailCabs drive without mechanical coupling. The loose coupling enables RailCabs to join or leave a convoy at full speed as shown in Fig. 2.1.7.

If a RailCab wants to join a convoy as a member, it needs to adapt its behavior for driving in the convoy by performing software reconfiguration. First, it needs to instantiate a software component that implements the communication protocols that are necessary for the convoy mode. Second, it needs to switch to a different velocity controller. If a RailCab drives alone or as a coordinator, the speed of the RailCab is controlled based on the current speed and the reference speed. If a RailCab is a member, it needs to switch to a velocity controller that also considers the current distance to the preceding RailCab in the convoy. Finally, the aforementioned software component and the new velocity controller need to be connected because the RailCab will now receive the reference speed from the coordinator of the convoy and must not decide on its speed itself.

To guarantee the correctness of the reconfiguration behavior and the communication protocols, we apply formal verification. The intention of the formal verification is to proof that the RailCab will have a correct software architecture at any given time, e.g. that a RailCab will only enter the convoy if it is approved by the coordinator, and that the communication protocol satisfies the required safety properties as illustrated in Sect. 5.2. Our formal verification includes the verification of timing properties as well [3], because the system has to react in time. In case of an emergency brake for example, each convoy member needs to be notified in time to avoid collisions. A particular difficulty for the verification is that the number of RailCabs participating in the convoy may vary during runtime.

## 2.2 Miniature Robot BeBot

Alexander Jungmann, Bernd Kleinjohann, Lisa Kleinjohann, Christoph Rasche, and Thomas Schierbaum

The miniature robot BeBot was designed as research platform to design and evaluate swarm intelligence algorithms, dynamic reconfiguration and multi-agent systems [11, 14]. Furthermore it is also a test bed for the technology "**Molded Interconnect Devices**" (MID). MID-parts are three-dimensional plastic parts, manufactured by partially metalizing their surface. The housing of the robot is realized as such a MID-component. It comprises mechanical and electrical components.

The next sections will describe the hardware and software system of the BeBots. Additionally, extension modules which are used for specific tasks are introduced. The last section will show the step by step implementation of self-x properties to the BeBot.

### *2.2.1 Basic Vehicle*

The miniature robot BeBot has a size of approximately $9cm \times 9cm \times 9cm$. Its chassis uses the technology MID and has traces directly on the body surface which offers new possibilities for the synergistic integration of mechanics and electronics (cf. Fig. 2.18) [20]. This technology is used for mounting 12 infrared sensors, a microcontroller, several transistors and resistors for preprocessing directly on the robot chassis. The drive of the robot consists of a chain drive. Together with two 2 W DC gear motors with built-in encoders, this robot provides the robots with robust motion capabilities, even on slightly rough ground. Such a drive supports omni-directional movements, as it allows the BeBot to turn right on the spot. Two lithium-ion accumulators are used as power supply. One supplies the power to the overall system while the other one is dedicated to the motor drives.

The BeBot uses a modular concept of information processing with two board slots. The lower board provides basic functions and power supply. Additionally, it contains several sensors, like a three axis acceleration sensor, a yaw rate gyroscope, and a sensor for monitoring the charging level of the battery. The upper board is responsible for wireless communication based on Zig-Bee, Bluetooth, and W-LAN. It is equipped with a low power system on chip, with package on package memory and provides an ARM Cortex-A8 600 MHz high performance processor, a TI C64x+ digital signal processor, 256 MB main memory and 512 MB flash memory. Additionally, the board also stores four groups of three high brightness tricolor LEDs. LEDs, allowing the BeBots to express their internal state via different colors. The robots are equipped with a camera and twelve infrared sensors, used for distance measurement and environment recognition. Based on this equipment, it is possible to perform complex calculations for determining the robot's behavior, e.g. image processing algorithms directly on the BeBot. Different techniques for energy saving, like dynamic frequency and voltage scaling as well as dynamic power down of unused hardware components, including RF processing, provide powerful computation capabilities with long battery life. Additionally, I2C, UART, USB, SDcard, camera, and memory interfaces as well as a small module slot, provide great expansion capabilities. On top of the body a light guide realized by a satined hemisphere and a cover plate with integrated WLAN-Antenna are installed. An infrared communication interface allows the equipping of the BeBots with mechanical extension modules wirelessly.

### *2.2.2 Extension Modules*

In order to make the BeBot universally applicable, three extension modules have been designed to allow the BeBots to handle different items [11]. These modules enhance the range of functions of the BeBots as described in the following.

The first module is the **grafter** shown in Fig. 2.19 (a). Its main task is to push different objects in a controlled manner. Objects can be stored within the graft unit allowing controlled object guidance, which is advantageous for a controlled depositing. Furthermore, slopes can be passed without losing objects located in the unit.

The second extension module is the **lifter**, which is built similarly to a forklift as depicted in Fig. 2.19 (b). Its task is to lift different items and, for instance, load them onto the transporter module described below. The gripping unit can be

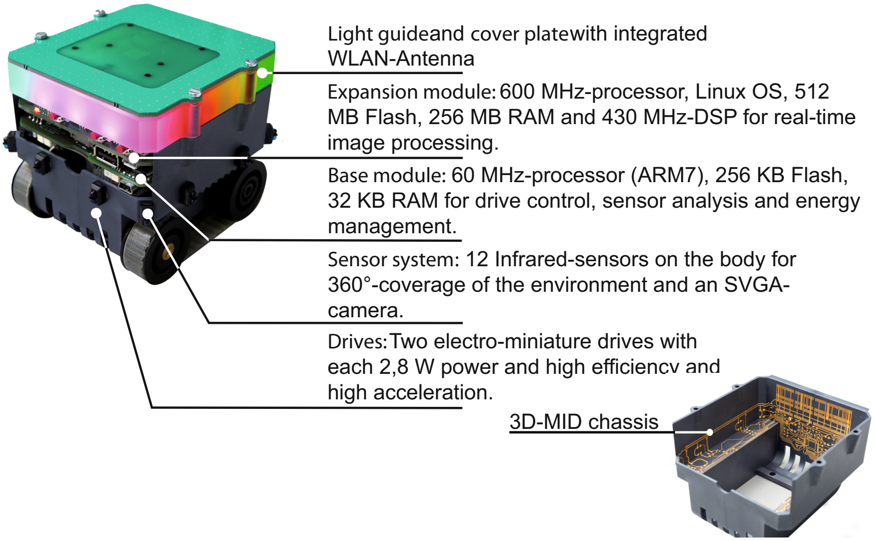
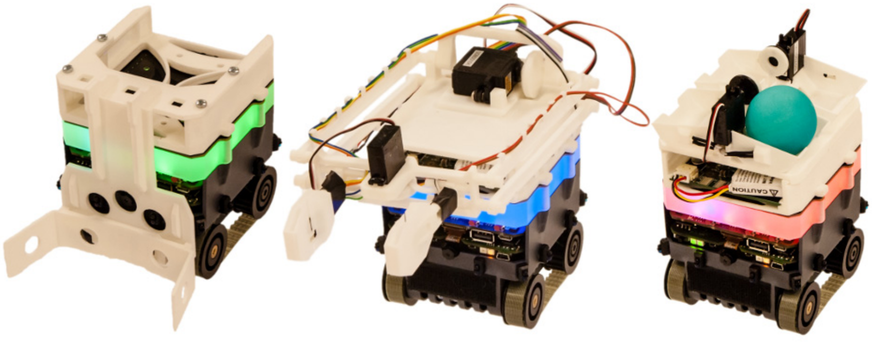

Light guideand cover platewith integrated WLAN-Antenna
Expansion module: 600 MHz-processor, Linux OS, 512 MB Flash, 256 MB RAM and 430 MHz-DSP for real-time image processing.
Base module: 60 MHz-processor (ARM7), 256 KB Flash, 32 KB RAM for drive control, sensor analysis and energy management.
Sensor system: 12 Infrared-sensors on the body for 360°-coverage of the environment and an SVGA-camera.
Drives: Two electro-miniature drives with each 2,8 W power and high efficiency and high acceleration.
3D-MID chassis

**Fig. 2.18** The miniature robot BeBot. On the left-hand side the two boards of the robot are depicted while on the right-hand side the MID technology is shown.

moved vertically. This is achieved by a gear rack attached to a gear wheel. The gear wheel is attached to a gear box, which is coupled to an electric engine. The gripping mechanism is also realized with an electric engine. The engine is coupled with a gear wheel, which in turn is attached to the grippers. By rotating the gear wheel the grippers are moved horizontally. An infrared sensor ensures that an object located between the gripping unit can be lifted correctly. Furthermore, the gripping unit is equipped with pressure sensors to control the gripping power generated by the electric engine. The lifter is controlled via a separate conductor board. The power supply is realized by a rechargeable battery.

The third extension module is the **transporter**. It is able to store up to four objects in separate cavities (cf. Fig. 2.19 (c)). Each object can be unloaded individually with the help of two electric drives, each serving two cavities. Objects located in a single cavity can be detected via infrared sensors installed on each side of the cavity. According to the modular assembly of the BeBot the transporter is also controlled via the same separate conductor board and rechargeable battery like the lifter.

The conductor board provides four half bridges to control two electric drives or four solenoids, four LED drivers and analog inputs to detect objects with infrared sensors and four general purpose, inputs to interpret two motor encoders. The communication between the BeBot and the extension module is realized by an infrared communication interface. If the extension module is detected and identified by the BeBot, it sets different colors at the light guide: green for the grafter, blue for the lifter and red for the transporter.



a) grafter b) lifter c) transporter

**Fig. 2.19** BeBots equipped with the extension modules

### *2.2.3 Operating System*

To execute programs directly on the robots, the BeBots provide the Linux distribution OpenRobotix [25]. It is based on a modified Linux kernel 3.0.32. The standard GNU C library is included and udev is used as device manager. Several standard GNU and Unix tools are available by the use of BusyBox, which provides a fairly complete software environment for any small and embedded system. The system can be built via an extension of the OpenEmbedded development environment, called OpenRobotix [25]. It allows the creation of an embedded Linux operating system. OpenRobotix enables the cross compilation of software packages for the use of the ARM processor of the BeBot. The existing software branch was extended to contain the robot specific information, patches and additional software, like the Player network server and drivers for the robot hardware.

The Linux kernel supports the WiFi device and all standard communication protocols. By using Linux BlueZ protocol stack, the Bluetooth communication, and all standard Bluetooth protocols like RFCOMM and BNEP are supported. Additionally, all Linux and platform-independent or ARM compatible protocols can be ported to the robot platform. One example is the ad hoc wireless mesh routing daemon OLSRD. It implements the optimized link state routing protocol and allows mesh routing on any network device.

In addition, system software concepts for self-optimizing applications were developed, as introduced in detail in Sect. 5.5. The real-time operating system ORCOS in contrast to Linux provides guaranteed real-time behavior. It has been ported to the BeBot as well.

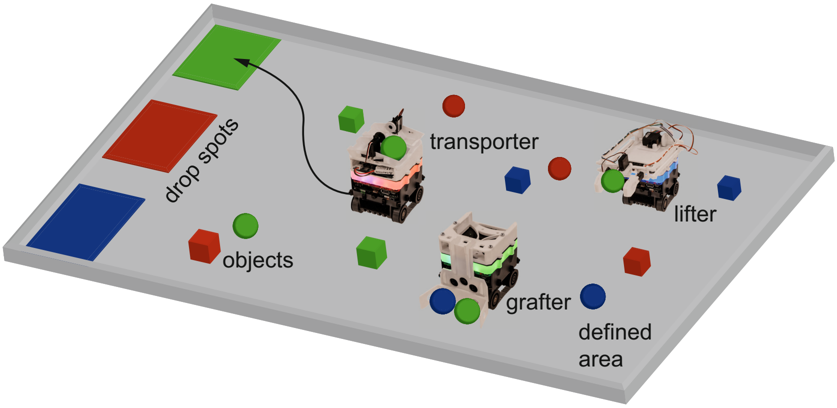


**Fig. 2.20** Schematic overview of the application scenario

### *2.2.4 Implementing Self-X Properties*

To investigate and evaluate algorithms that enable an entire group of BeBots to optimize its cooperative behavior while simultaneously taking environment changes into account, a complex application scenario was designed: Objects of different color and geometric shape are randomly distributed throughout the entire playground. Smaller sections (drop spots) with the same color of the existing objects are additionally placed inside the area (cf. Fig. 2.20). The positions of the objects as well as the drop spots are not known in advance, but have to be found by the group of BeBots. The main collective task for the group of BeBots is to locate all of those objects and bring them to the drop spots with the corresponding color.

In consideration of the described scenario, the integration of self-optimization seems obvious: it is nearly impossible to take all situations, which may appear during a real world scenario, into account, while implementing the behavior of a single BeBot. Therefore, a BeBot must posess the ability to adapt its behavior to unforeseen situations and changes in the environment (cf. Sect. 1.2). When incorporating objectives like "collect the objects as fast as possible" or "expend the lowest possible effort while sticking to a given deadline" occur, we are talking about external objectives that have to be optimized during operation mode. With respect to the two mentioned objectives, the BeBots should self-optimize their behavior in order to minimize the required time for collecting all objects and to minimize the effort in terms of the overall power consumption, respectively. Furthermore, the robots apparently have to involve **self-organization** mechanisms in order to apply appropriate team coordination for accomplishing the current goal as a collective. To be able to perform the described scenario in a self-optimizing manner, different properties were developed. The main steps will be explained in the following.

To allow for deliberative behavior, first of all a single BeBot has to incorporate means for realizing **self-awareness**: a BeBot not only has to react to sensor information, but to maintain a model of its current state as well as its environment in

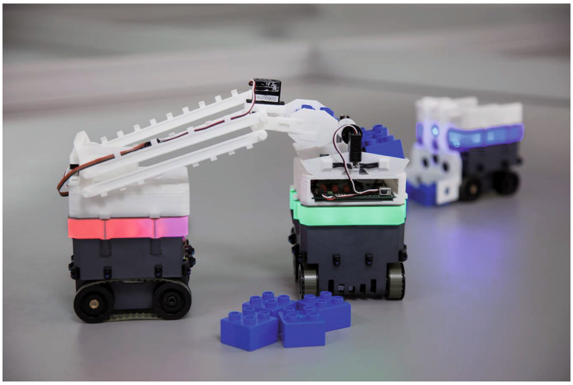


**Fig. 2.21** Cooperation of the BeBots in the application scenario

order to analyze the current situation and create a plan for future actions. Within the scope of an initial application scenario, in which a BeBot autonomously has to fulfill the external objective of driving slalom through a chain of small traffic cones [17], a light-weight architecture incorporating a reactive component based on Motor Schemes [25] for reactive behavior, a deliberative component for planning behavior based on Hierarchical State Machines as well as an efficient color-based image processing approach was realized [19]. Based on the processed and abstracted image data, the BeBot is able to reason about its current situation and to adapt its behavior according to its currently active internal objective such as passing the next traffic cone either on the left side or on the right side.

In a next step, the extension modules were integrated into a separate application scenario, where a BeBot equipped with a lifter module had to cooperate with a BeBot equipped with a transporter module in order to load detected items (external objective) (cf. Fig. 2.21). For enabling transporter and lifter to cooperate with each other during the loading process, a message based communication component was realized. To enable the lifter to successfully load an item into a cavity of the transporter, the light guide of the transporter was used to indicate free cavities. Based on the image processing approach, the color code indicated by the transporters light guide was decoded and used by the lifter to identify an empty cavity and to correctly align to the cavity for the loading process. In terms of the self-optimization process, the BeBots' capabilities of analyzing their current situation were extended in order to determine their situation dependent, internal objective such as "searching for objects" or "loading objects onto the transporter". In a final step, the BeBots' behavior for the initially described scenario was realized by combining all developed components. To separate the localization problem from the situation analyzing phase, the entire scenario took place in a test bed [18] that is able to determine each BeBot's position on the playground based on unique landmarks. By means of the image processing approach, items as well as team mates can be detected by a single BeBot. Reactive behavior enables a BeBot to avoid obstacles (inherent objective). The more complex deliberative component allows for planning and decision-making in order to change strategy, depending on the current game situation [27]. In order to enable a BeBot to adapt to environmental changes and to improve its individual behavior, a statistical planner based on Reinforcement Learning was developed (cf. Sect. 5.3.9). Organization between team mates is realized based on a distributed communication component.

There exist still a lot of possibilities for improving the BeBot behavior. In fact, machine learning techniques can be applied to most of the mentioned components in order to improve the BeBot's capability of adapting its behavior according to unpredictable situations. Nevertheless, the BeBot with all its hardware capabilities is a decent platform for developing and investigating algorithms that realize self-x properties under real-world conditions. The entire self-optimization process can be seamlessly incorporated into the BeBot based on the conceptual design described in Chap. 4.

## 2.3 X-by-Wire Test Vehicle

Peter Reinold and Ansgar Trächtler

As an example to demonstrate a self-optimizing system, the fully active mechatronic test vehicle **Chameleon** was developed (cf. Fig. 2.22). It is actuated entirely electronically. The necessary energy is provided by a lithium-ion accumulator. It can carry one person (the driver) and is controlled via a joystick. The Chameleon is an X-by-wire vehicle i.e. there are no mechanical couplings between the control element (in our case the joystick) and the actuators. A rapid prototyping hardware is used as the control unit. The empty weight is approx. 280 $kg$ and the maximum speed is about 50 $km/h$.

It is built up modularly and has four identically constructed corner modules with three DC motors each: One for steering, one for driving and one for an active suspension. They enable

- all-wheel drive
- single-wheel steering
- active suspension.

Thus all relevant degrees of freedom of the wheel (except for the camber angle) can be influenced. This multitude of possible interventions enables the influence of longitudinal and lateral dynamics systematically. The system is overactuated: Thus there are degrees of freedom to realize the same global movement.

The driving motors are used to drive and brake the vehicle. In this process the driving motors act as generators and make recovering energy possible. Additionally the single-wheel steering enables the deceleration of the vehicle by turning the wheels inwards. The mechanical brake is used only in the case of an emergency.

Even a cornering manoeuvre can be realized in several ways:

- conventional front-wheel steering,
- all-wheel steering,
- different driving torques/wheel speeds on the left- and right-hand sides (torque vectoring),
- a combination of all-wheel-steering and torque vectoring.

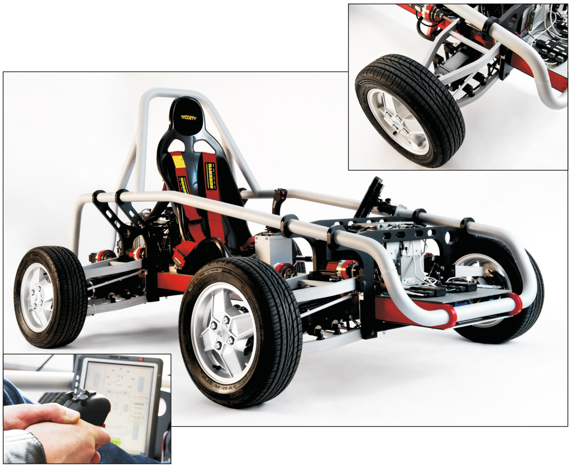


**Fig. 2.22** Test vehicle Chameleon

## 2.3.1 *Vehicle Dynamics*

The degrees of freedom of the vehicle's movement can be allocated to different domains of the vehicle dynamics: the vertical, the longitudinal and the lateral dynamics. The combination of the first two is called the horizontal dynamics which describe the planar movement of the vehicle. It can be described by three degrees of freedom: the yaw rate $\dot{\psi}$ (rotational speed around the vertical axis), the velocity $v$ and the slip angle $\beta$ (angle between the longitudinal direction of the vehicle and the velocity vector). The resulting longitudinal and lateral forces at the center of gravity of the vehicle $F_{x,V}$ and $F_{y,V}$ and the yaw moment $M_{z,V}$ can be computed unambiguously by the three kinematic values $v$, $\beta$, and $\dot{\psi}$ and their temporal derivatives with the inverse dynamics:

$$F_{x,V} = m_V(\dot{v} \cdot \cos\beta - v(\dot{\psi} + \dot{\beta}) \cdot \sin\beta) \tag{2.7}$$

$$F_{y,V} = m_V(\dot{v} \cdot \sin\beta + v(\dot{\psi} + \dot{\beta}) \cdot \cos\beta) \tag{2.8}$$

$$M_{z,V} = J_{zV} \cdot \ddot{\psi} \tag{2.9}$$

$m_V$ is the mass of the vehicle and $J_{z,V}$ is its moment of inertia about the yaw movement.

As each tire has a longitudinal force $F_{x,i}$ and a lateral force $F_{y,i}$, there are in total the eight horizontal tire forces which can be influenced by the driving and steering motors. The index $i$=1..4 represents the wheel. To compute these tire forces the well-known Pacejka tire model is used [26]. Among other things the tire forces depend on the longitudinal tire slip $\lambda_i$ and the slip angle $\alpha_i$[4]. These eight forces result in three

[4] The slip angle $\alpha_i$ is the angle between the tire velocity and its longitudinal axis.

forces/moments at the center of gravity ($F_{x,V}, F_{y,V}, M_{z,V}$). Thus there are degrees of freedom for the force distribution. With $\delta_i$ to be the steering angle of the different wheels, one gets:

$$F_{x,V} = f(F_{x,i}, F_{y,i}, \delta_i) \tag{2.10}$$

$$F_{y,V} = f(F_{x,i}, F_{y,i}, \delta_i) \tag{2.11}$$

$$M_{z,V} = f(F_{x,i}, F_{y,i}, \delta_i) \tag{2.12}$$

The vertical dynamics can be used to improve comfort and safety independent from the horizontal dynamics control or to influence the horizontal dynamics systematically by changing the wheel loads. Thus there are degrees of freedom for the distribution of the forces to the tires.

### 2.3.2 Self-optimizing Integrated Vehicle-Dynamics Control

One aim of the CRC was to develop a self-optimizing vehicle-dynamics control, which uses degrees of freedom that were described in the section above for an optimization to ensure an optimal realization of a desired movement in changing environmental conditions. E.g. in case of a low state of charge of the battery it is important to save energy. In other driving situations the tire wear may be important. For safety reasons the utilization of the adhesion potential has to be minimized. As these objectives are contradictory, a compromise between them has to be found.

A simplified structure for the self-optimizing vehicle-dynamics control is presented in Fig. 2.23. First the reference values for the velocity $v$, the slip angle $\beta$ and the yaw rate $\dot{\psi}$ are generated based on the driver's input. As the driver gives only two input values with the joystick (the longitudinal deflection $x_j$ and the lateral deflection $y_j$) but three kinematic values to be computed, there is a degree of freedom in this set-point generation. Based on these set points, the desired forces and torques at the center of gravity can be computed with the inverse dynamics as presented in Eqs. 2.7-2.9. A model-based optimization distributes theses forces at the center of gravity to the tires and computes the optimal values for the tire slips $\lambda_i$ and the tire slip angles $\alpha_i$ for the four wheels. Also the vertical dynamics can be considered in this optimization. The optimized values of $\lambda_i$ and $\alpha_i$ are used as reference values for the local controllers of different actuators. Single-wheel actuation enables the influencing of $\lambda_i$ and $\alpha_i$ systematically by using the driving motors and steering motors.

The optimization objectives of the vehicle-dynamics control are [29]:

- Minimization of the tire wear by minimizing the tire slip angles $\alpha_i$:

$$\min_{\alpha_i} \left( f_1 = \sum_{i=1..4} |\alpha_i| \right) \tag{2.13}$$

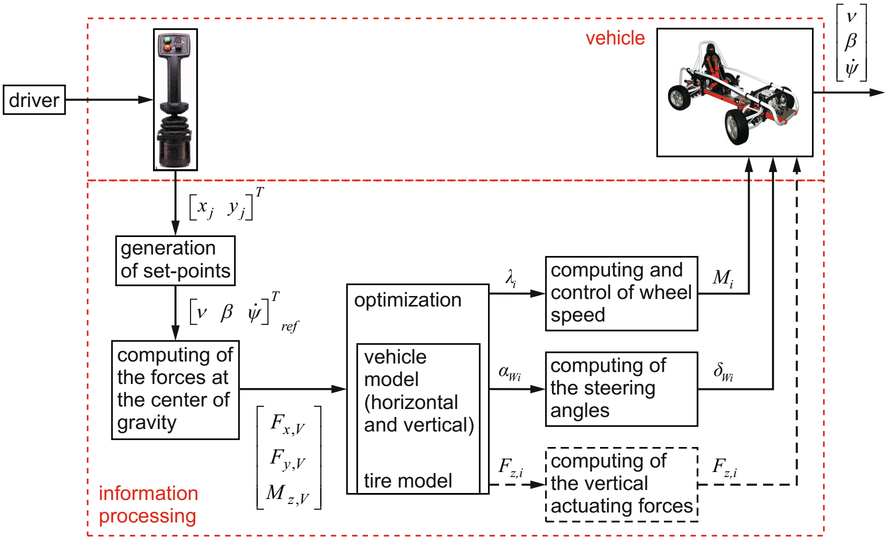

vehicle
driver
$[v\ \beta\ \dot{\psi}]$
$[x_j\ y_j]^T$
generation of set-points
$[v\ \beta\ \dot{\psi}]^T_{ref}$
computing of the forces at the center of gravity
$F_{x,V}$ $F_{y,V}$ $M_{z,V}$
optimization
vehicle model (horizontal and vertical)
tire model
$\lambda_i$
computing and control of wheel speed
$M_i$
$\alpha_{Wi}$
computing of the steering angles
$\delta_{Wi}$
$F_{z,i}$
computing of the vertical actuating forces
$F_{z,i}$
information processing

**Fig. 2.23** Control structure without closed-loop control [29]

- Minimization of the energy consumption by minimizing the longitudinal tire forces[5]:

$$\min_{\alpha_i,\lambda_i}\left(f_2=\sum_{i=1..4}F_{x,i}\right) \tag{2.14}$$

- Minimization of the maximum utilization of the adhesion potential represented by the ratio between the magnitude of the tire force and its adhesion limit:

$$\min_{\alpha_i,\lambda_i}\left(f_3=\max_{i=1..4}\frac{\sqrt{F_{x,i}^2+F_{y,i}^2}}{\mu\cdot F_{z,i}}\right) \tag{2.15}$$

The purpose of the third objective is to increase the systems reliability.

As the vehicle has to perform its driving task, the realization of the desired forces are regarded as constraints. Thus the realization of the desired forces/torques at the center of gravity have to be realized by the tire forces (cf. Eqs. 2.10-2.12). Even the technical limitations of the driving torques $M_i$ and the steering angles $\delta_i$ have to be taken into account as constraints. The physical limitation of the tire forces is already considered in the tire model, which is the basis of the optimization. Hence it doesn't need to be considered as an additional constraint.

[5] In the case of braking negative longitudinal tire forces can be used for energy recovery.

Furthermore it is possible to consider actuator breakdowns or to eliminate degrees of freedom which do not exist by using additional constraints. Thus the optimization strategy can be adapted to other vehicles and actuator concepts [2], [28], [39]. Additional constraints can be included e.g. to prevent actuators from damage.

The three objectives (Eqs. 2.13-2.15) are contradictory - depending on the driving situation. The multiobjective optimization problem is solved as a weighted sum of the objectives:

$$\min_{\alpha_i,\lambda_i}\left(f=\sum_{k=1}^{3} g_k\cdot f_k\right) \tag{2.16}$$

The optimization is used to compute reference values for the actuating variables. Depending on the situation the weighting factors $g_i$ are changed.

E.g. if the battery's state of charge low, the objective to minimize the energy consumption becomes more important. This is the basis for a self-optimizing vehicle-dynamics control which changes the weighting of the objectives autonomously depending on the actual driving situation. Thus the behavior can be adapted automatically to changing environmental conditions.

In particular for braking maneuvers so-called set-oriented methods were used for the multiobjective optimization of the distribution of the tire forces. The applied technique is described in more detail in Sect. 5.3.1.

Due to model inaccuracies and disturbances differences between desired and actual movement are still possible. Thus a closed-loop controller is necessary. The driver of course can act as the controller and compensate for such differences. This is basically the same task a driver performs in conventional cars. But this would not tap the potentials of this control approach completely. The velocity $v$, the slip angle $\beta$ and yaw rate $\dot{\psi}$ have to be controlled. Thus the control strategy includes vehicle-dynamics control systems and driver assistance systems like electronic stability control (ESC) and cruise control. If we also use an additional feedback-control we should avoid adding values after the optimization because this would adulterate the optimized variables. Thus an outer feedback-control loop should not use the slip variables $\lambda_i$ or the slip angles $\alpha_i$ as actuating variables. This would eliminate the benefit of the optimization since the sum is no longer optimal concerning the used objectives. For this reason we consider the optimization as a part of the plant. This enhanced plant consists of the optimization, the local controllers and the vehicle itself (cf. Fig. 2.24). It has three input and three output values. In this case the enhanced plant is no longer over-actuated and the optimization results are not adulterated. With some simplifications the enhanced plant can be described as a nonlinear control-affine state space model. Hence we can use exact linearization techniques. The nonlinear decoupling feed-forward controller $M_{lin}(x)$ and the feed-back controller $r_{lin}(x)$ are computed in the way described detailed in [16], [9] and [21]. The resulting overall behavior of the closed-loop control system consists of three decoupled first-order lag elements. In addition, an outer control loop, a conventional 2-degree-of-freedom structure is used. Due to the decoupling and linearization of the plant, the controller $C$ of the outer loop can be designed with the methods used for LTI-SISO-systems. The resulting structure is presented in Fig. 2.24.

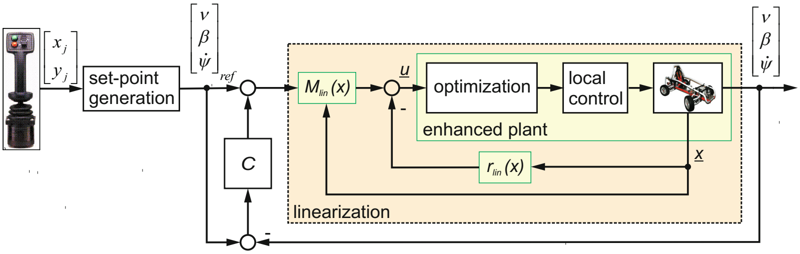


**Fig. 2.24** Structure with control loop with the enhanced plant, the linearization and the feedback-control loop

As a consequence of this control structure the driver is decoupled from the actuating variables of the real actuators. E.g. the driver has no direct influence on the four steering angles $\delta_i$. The driver describes how the vehicle should move, but not how this should be realized.

By using self-optimizing techniques a multitude of different and contradictory objectives can be achieved. An optimal compromise for the current driving situation can be found instead of a fixed compromise which may not be optimal for each specific situation.

## References

1. VDI 2057, Part 1: Human Exposure to Mechanical Vibrations - Whole-body Vibration (2004)
2. Andreasson, J., Knobel, C., Buente, T.: On Road Vehicle Motion Control - Striving Towards Synergy. In: Proceedings of the 8th International Symposium on Advanced Vehicle Control, Taipei (2006)
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Böcker, J., Schulz, B., Knoke, T., Fröhleke, N.: Self-Optimization as a Framework for Advanced Control Systems. In: Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics, Paris (2006)
5. Ettingshausen, C., Hestermeyer, T., Otto, S.: Aktive Spurführung und Lenkung von Schienenfahrzeugen. In: Tagungsband der 6. Magdeburger Maschinenbautage, Intelligente technische Systeme und Prozesse. Grundlagen, Marburg (2003)
6. Flaßkamp, K., Heinzemann, C., Krüger, M., Steenken, D., Ober-Blöbaum, S., Schäfer, W., Trachtler, A., Wehrheim, H.: Sichere Konvoibildung mit Hilfe optimaler Bremsprofile. In: Gausemeier, J., Rammig, F.J., Schäfer, W., Trachtler, A. (eds.) Tagungsband Zum 9. Paderborner Workshop Entwurf mechatronischer Systeme, HNI-Verlagsschriftenreihe, Paderborn (2013)
7. Flaßkamp, K., Ober-Blöbaum, S., Ringkamp, M., Schneider, T., Schulte, C., Böcker, J.: Berechnung optimaler Stromprofile für einen 6-phasigen, geschalteten Reluktanzantrieb. In: Tagungsband Vom 8, Paderborn. Paderborner Workshop Entwurf mechatronischer Systeme. Heinz Nixdorf Institut Verlagsschriftreihe, Paderborn (2011)

8. Flaßkamp, K., Ober-Blöbaum, S., Schneider, T., Böcker, J.: Optimal Control of a Switched Reluctance Drive by a Direct Method Using a Discrete Variational Principle. In: Proceedings of the 52nd IEEE Conference on Decision and Control, Florenz (2013)
9. Foellinger, O.: Nichtlineare Regelungen II, 7th edn. Oldenbourg Verlag, München (1993)
10. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
11. Gausemeier, J., Schierbaum, T., Dumitrescu, R., Herbrechtsmeier, S., Jungmann, A.: Miniature Robot BeBot - Mechatronic Test Platform for Self-x Properties. In: Proceedings of the 9th IEEE International Conference on Industrial Informatics, Lisbon, pp. 451–456 (2011)
12. Geisler, J., Witting, K., Trächtler, A., Dellnitz, M.: Multiobjective Optimization of Control Trajectories for the Guidance of a Rail-bound Vehicle. In: Proceedings of the 17th IFAC World Congress, Seoul (2008)
13. Henke, C., Tichy, M., Schneider, T., Böcker, J., Schäfer, W.: Organization and Control of Autonomous Railway Convoys. In: Proceedings of the 9th International Symposium on Advanced Vehicle Control, Kobe, pp. 318–323 (2008)
14. Herbrechtsmeier, S., Witkowski, U., Rückert, U.: BeBot - A Modular Mobile Miniature Robot Platform Supporting Hardware Reconfiguration and Multi-standard Communication. In: Kim, J.-H., Ge, S.S., Vadakkepat, P., Jesse, N., Al Manum, A., Puthusserypady, K.S., Rückert, U., Sitte, J., Witkowski, U., Nakatsu, R., Braunl, T., Baltes, J., Anderson, J., Wong, C.-C., Verner, I., Ahlgren, D. (eds.) Progress in Robotics. CCIS, vol. 44, pp. 346–356. Springer, Heidelberg (2009)
15. Hölscher, C., Keßler, J.H., Krüger, M., Trächtler, A., Zimmer, D.: Hierarchical Optimization of Coupled Self-optimizing Systems. In: Proceedings of the 10th IEEE International Conference on Industrial Informatics, Beijing (2012)
16. Isidori, A.: Nonlinear Control Systems, 2nd edn. Springer, Heidelberg (1989)
17. Jungmann, A., Kleinjohann, B., Kleinjohann, L., Bieshaar, M.: Efficient Color-Based Image Segmentation and Feature Classification for Image Processing in Embedded Systems. In: Proceedings of the 4th International Conference on Resource Intensive Applications and Services, St. Maarten (2012)
18. Jungmann, A., Lutterbeck, J., Werdehausen, B., Kleinjohann, B.: A Test Bed for Investigating Self-X Properties in Multi-Robot Societies. In: Proceedings of the 9th IEEE International Conference on Industrial Informatics, Lisbon, pp. 437–442 (2011)
19. Jungmann, A., Schierbaum, T., Kleinjohann, B.: Image Segmentation for Object Detection on a Deeply Embedded Miniature Robot. In: Proceedings of the Seventh International Conference on Computer Vision Theory and Applications (VISAPP), Rome, pp. 441–444. Insticc Press, Setubal (2012)
20. Kaiser, I., Kaulmann, T., Gausemeier, J., Witkowski, U.: Miniaturization of Autonomous Robot by the new Technology Molded Interconnect Devices (MID). In: Proceedings of the 4th International AMiRE Symposium, Buenos Aires (2007)
21. Khalil, H.K.: Nonlinear Systems, 2nd edn. Prentice-Hall, New Jersey (1996)
22. Knoke, T.: Entwurf und Betrieb hybrid-elektrischer Fahrzeugantriebe am Beispiel von Abfallsammelfahrzeugen. Ph.D. thesis, Institute for Power Electronics and Electrical Drives, Universität Paderborn (2010)
23. Krüger, M., Remirez, A., Keßler, J.H., Trächtler, A.: Discrete Objective-based Control for Self-Optimizing Systems. In: Proceedings of the American Control Conference, Washington (2013)

24. Münch, E.: Selbstoptimierung verteilter mechatronischer Systeme auf Basis paretooptimaler Systemkonfigurationen. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagschriftenreihe, Paderborn (2012)
25. Openrobotix: Open Embedded Based Open Source Linux Distribution for Mini Robots (2012), http://openrobotix.berlios.de/ (accessed September 10, 2012)
26. Pacejka, H.B.: Tyre and Vehicle Dynamics, 2nd edn. Butterworth-Heinemann, Amsterdam (2006)
27. Rasche, C., Jungmann, A., Schierbaum, T., Werdehausen, B., Kleinjohann, B.: Towards Hierarchical Self-Optimization in Autonomous Groups of Mobile Robots. In: Proceedings of the 10th International Conference on Industrial Informatics, Beijing (2012)
28. Reinold, P., Nachtigal, V., Trächtler, A.: An Advanced Electric Vehicle for the Development and Test of New Vehicle-Dynamics Control Strategies. In: Proceedings of the 6th IFAC Symposium on Advances in Automotive Control AAC, München (2010)
29. Reinold, P., Traechtler, A.: Multi-objective Optimization for the Determination of the Actuating Variables of the Horizontal Dynamics of an Electric Vehicle with Single-wheel Chassis Actuators. In: Tagungsband vom Autoreg: Steuerung und Regelung von Fahrzeugen und Motoren, Baden-Baden, pp. 185–198 (2011)
30. Romaus, C.: Selbstoptimierende Betriebsstrategien für ein hybrides Energiespeichersystem aus Batterien und Doppelschichtkondensatoren. Ph.D. thesis, Berichte aus dem Fachgebiet Leistungselektronik und Elektrische Antriebstechnik Band 3, Shaker-Verlag, Aachen (2013)
31. Romaus, C., Bocker, J., Witting, K., Seifried, A., Znamenshchykov, O.: Optimal Energy Management for a Hybrid Energy Storage System Combining Batteries and Double Layer Capacitors. In: Proceedings of the Energy Conversion Congress and Exposition, San Jose, pp. 1640–1647 (2009)
32. Romaus, C., Gathmann, K., Böcker, J.: Optimal Energy Management for a Hybrid Energy Storage System for Electric Vehicles Based on Stochastic Dynamic Programming. In: Proceedings of the Vehicle Power and Propulsion Conference, Lille (2010)
33. Schneider, T.: Traktionsantrieb mit linearem, geschalteten Reluktanzmotor für ein autonomes Bahnfahrzeug. Ph.D. thesis, Universität Paderborn
34. Schneider, T., Schulte, C., Mathapati, S., Böcker, J.: Energy Transfer with Doubly-excited Switched Reluctance Drive. In: Proceedings of the International Symposium on Power Electronics, Electrical Drives, Automation and Motion, Pisa (2010)
35. Schneider, T., Schulz, B., Henke, C., Witting, K., Steenken, D., Böcker, J.: Energy Transfer via Linear Doubly-fed Motor in Different Operating Modes. In: Proceedings of the International Electric Machines and Drives Conference, Miami, pp. 598–605 (2009)
36. Schulz, B.: Selbstoptimierende Antriebsregelung. Ph.D. thesis, Universität Paderborn
37. Schulz, B., Pottharst, A., Fröhleke, N., Böcker, J.: Modelling of Influences to a Linear-Drive-System. In: Proceedings of the 11th Int. Power Electronics and Motion Control Conferences, Riga (2004)
38. Sondermann-Wölke, C., Sextro, W.: Integration of Condition Monitoring in Self-Optimizing Function Modules Applied to the Active Railway Guidance Module. International Journal on Advances in Intelligent Systems 3(1&2), 65–74 (2010)
39. Sondermann-Wölke, C., Sextro, W., Reinold, P., Traechtler, A.: Reliability-oriented Multi-objective Optimization for the Actuator Reconfiguration of an X-by-wire Vehicle. In: Tagungsband Technische Zuverlaessigkeit, Leonberg (2011)
40. Stille, K.S., Romaus, C., Böcker, J.: Online Capable Optimized Planning of Power Split in a Hybrid Energy Storage System. In: 25th International Conference on Computer as a Tool, Zagreb (2013)

41. Walther, M., Müller, T., Wallaschek, J.: Optimisation of Mechatronic Systems Using Dependability Oriented Design Methods. In: Proceedings of the Mechatronic Systems and Materials, Cracow (2006)
42. Witting, K., Schulz, B., Dellnitz, M., Böcker, J., Fröhleke, N.: A new Approach for Online Multiobjective Optimization of Mechatronic Systems. International Journal on Software Tools for Technology Transfer STTT 10(3), 223–231 (2008)

# Chapter 3
# Development of Self-optimizing Systems

Jürgen Gausemeier, Sebastian Korf, Mario Porrmann, Katharina Stahl,
Oliver Sudmann, and Mareen Vaßholz

**Abstract.** The development of self-optimizing systems is challenging due to the involvement of different domains, such as mechanical, electrical/electronic, control and software engineering as well as experts from higher mathematics and artificial intelligence. This leads to an increased design complexity and requires an effective communication and cooperation between the developers. The Collaborative Research Centre 614 developed a design methodology for self-optimizing systems that extends existing methodologies for mechatronic systems to support the developer appropriately with self-optimization specific expertise. The methodology consists of a reference process, methods and tools. The reference process is based on our experiences from the development of the RailCab and its function modules and is recommendatory. It shows the ideal approach for the development of self-optimizing systems. It is structured into two main phases: "Domain-Spanning Conceptual Design" and "Domain-Specific Design and Development". The different steps in each phase are described in detail in this chapter. To make sure, that the necessary self-optimization expertise, especially from higher mathematics and artificial intelligence, is available during the development process, solution patterns are used. The specific methods for the development of self-optimizing systems are introduced in the following chapters of this book. Each development task has its own characteristics according to the project, the system tpye, and the environment of the development project, therefore an implementation model for the development process is needed, which is modeled individually with the components of the reference process. Furthermore, the planned process sequence of the implementation model can vary due to changing development objectives such as time and costs during the project excecution. Our framework of a self-optimizing development process supports the management of the development process.

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*,
Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_3, © Springer-Verlag Berlin Heidelberg 2014

## 3.1 Design Methodology for Self-optimizing Systems

Jürgen Gausemeier and Mareen Vaßholz

The previous chapter has shown the benefit of the functionality of self-optimizing systems. But also that this leads to an increasing design complexity and requires an effective communication and cooperation between developers from different domains throughout the development process. In addition to mechanical, electrical/-electronic, control and software engineers, experts from advanced mathematics and artificial intelligence are involved. This requires a fundamental understanding of the entire system as well as of its development.

The Collaborative Research Center (CRC) 614 pursued the long-term aim to open up the active paradigm of self-optimization for mechanical engineering and to enable others to develop these systems. Therefore a design methodology was developed, that expands existing ones for mechatronic systems such as the VDI guideline 2206 [1], the approach by Isermann (2008) [32] or Ehrlenspiel (2007) [18], the iPeM-Modell [5] or the V-Model by Bender (2005) [10] and supports developers by providing domain-spanning and self-optimization-specific methods and tools [34]. The methodology consists of a reference process, methods and tools for the development of self-optimizing systems. The **reference process** for self-optimizing systems was developed based on our experiences in the development of the RailCab and its function modules (cf. Sect. 2.1). It describes the chronological sequence with regard to the content of the development process for self-optimizing systems based on the modeling language OMEGA. The reference process provides guide lines on how to apply self-optimization methods and solution pattern for self-optimizing systems, that bind the knowledge of self-optimization experts. The reference process serves as basis for the application to a specific development task and company in form of an **implementation model**. This model provides a detailed process sequence according to the project, the system type and the environment of the development project. It consists of process steps from the reference process and builds the starting point for the project management. For example the implementation model for the RailCab consists of about 850 steps and 900 development objects. During the development project execution, deviations and changes from the planned development process can occur, for example due to changing development objectives such as time and costs. To support the management during the project execution to react to these changes we adopted the self-optimizing approach to the management of the development process (cf. Sect. 3.4).

In accordance with the existing development methodologies, the development of self-optimizing systems can be basically structured into two main phases: the "Domain-Spanning Conceptual Design" and the "Domain-Specific Design and Development" (Fig.3.1). Within the conceptual design, the basic structure and the operation mode of the system are defined and the system is also structured into subsystems. This division into subsystems has to result in a development-oriented product structure, which integrates the two basic and mostly contradictory views of

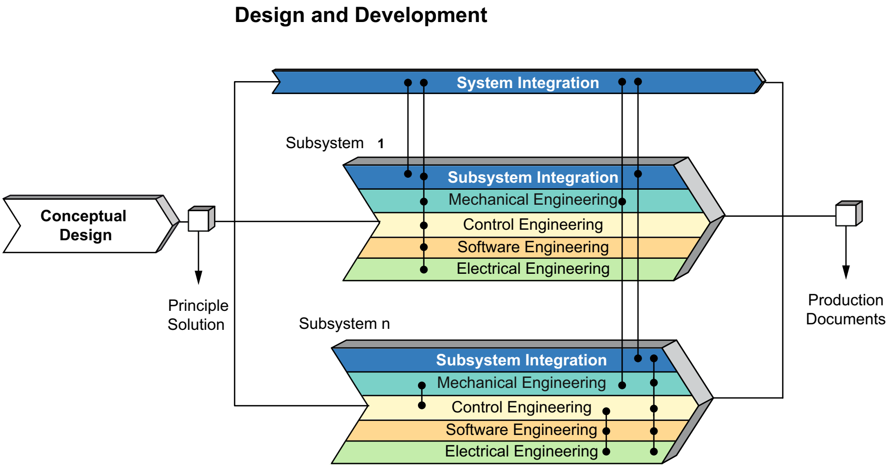


**Fig. 3.1** Macro cycle for the development of self-optimizing systems

shape- and function-oriented structure (cf. Sect. 4.6). This is a recursive process, which means that subsystems can also be assembled by further subsystems.

All the results of the conceptual design are specified in the so called principle solution. The principle solution based on the specification technique CONSENS (CONceptual design Specification technique for ENgineering of complex Systems, cf. Sect. 4.1), is built up to create common understanding of the system to be developed. The procedure used to develop the principle solution is described in Sect. 3.2. Based upon the principle solution, the subsequent domain-specific design and development is planned and realized. All defined subsystems are developed in parallel to each other and each subsystem is developed in parallel within the participating domains [34]. During this domain-specific phase, the domains involved, work in parallel with their domain-specific methods and tools, e.g. MCAD, ECAD or MATLAB. During the domain-specific design and development, system optimization is conducted in the domain control engineering as well, which particularly requires the involvement of experts from advanced mathematics and artificial intelligence. This specific knowledge is not available in development departments nowadays, therefore the expertise is made available by solution patterns (cf. Sect. 4.5). Because the domains work in parallel during the design and development it is necessary to ensure the consistency of the system model. Therefore the results of the domains are integrated continuously. For this purpose, model transformation and synchronization techniques are used (cf. Sect. 5.1). The integrated system is tested as a virtual prototype, to identify faults (cf. Sect. 5.6). This allows a short response time to failures and therefore reduces time and cost intensive iterations. The result of the design and development phase is the engineering data, such as manufacturing drawings or part lists [15].

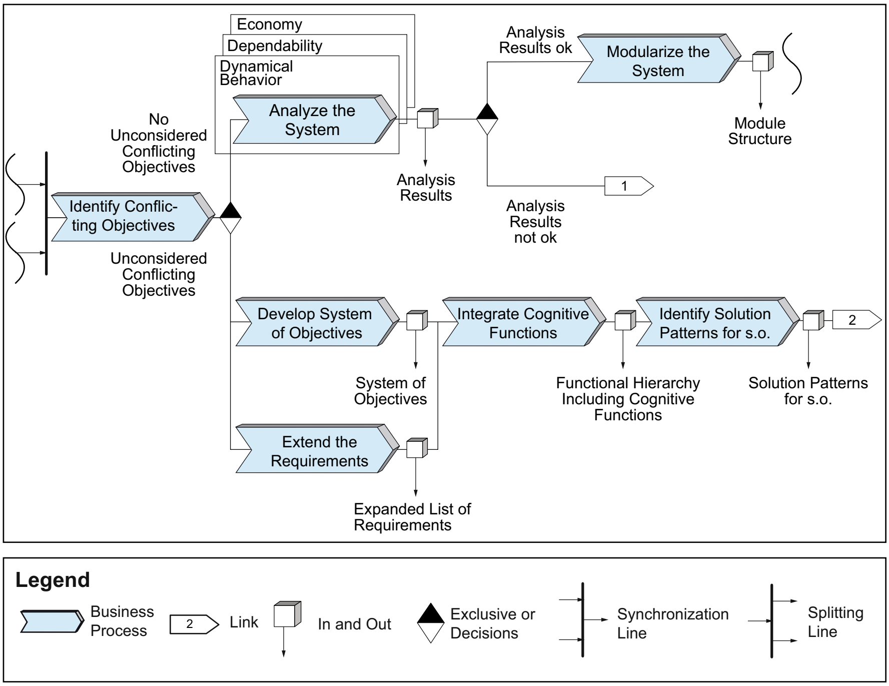

Economy
Dependability
Dynamical Behavior
Analyze the System
No Unconsidered Conflicting Objectives
Identify Conflicting Objectives
Unconsidered Conflicting Objectives
Analysis Results
Analysis Results ok
Modularize the System
Module Structure
1
Analysis Results not ok
Develop System of Objectives
System of Objectives
Integrate Cognitive Functions
Functional Hierarchy Including Cognitive Functions
Identify Solution Patterns for s.o.
Solution Patterns for s.o.
2
Extend the Requirements
Expanded List of Requirements
Legend
Business Process
2
Link
In and Out
Exclusive or Decisions
Synchronization Line
Splitting Line

**Fig. 3.2** Elements of the modeling language OMEGA

To serve as a compendium for developers to develop self-optimizing systems independently, this level of detail is not appropriate enough. For this reason, the macro cycle is refined by our reference process.

Therefore we use the object-oriented modeling language **OMEGA** (Object-Oriented[6] Method of Business Process Modeling and Analysis). It was developed at the Heinz Nixdorf Institute of the University of Paderborn for modeling and analyzing business processes. OMEGA allows the modeling of the entire operational structure of a company. It is also a good instrument for the analysis and planning of business processes, because of its intuitively understandable visualization with an easy and concise imagery [34]. For the description of the reference process we focus on the following elements of the modeling language (cf. Fig. 3.2):

Business process:

A business process is a sequence of activities to create an output or to transform an input. It has a defined starting point (trigger or input) and a defined endpoint (result or output).

[6] With respect to the development objects and not within the meaning of software engineering.

Process objects:
: Process objects are input and output variables of business processes. Usually a process object, which has been generated or transformed by a business process, is the input object of a subsequent process.

Connector:
: A connector can mark iterations within the development process. It can also be used to assign the results of one process step to another.

Synchronization/Splitting Line:
: This line synchronizes or splits the development tasks of different domains. At these points process objects can be exchanged.

Exclusives or Decisions:
: At this point different lines of actions are possible based on the state of the process object.

## 3.2 Domain-Spanning Conceptual Design

Jürgen Gausemeier and Mareen Vaßholz

The aim of the domain-spanning conceptual design is to develop the principle solution of the system. The **principle solution** describes the physical and the logical operating characteristics of the system. The description of the principle solution is structured into the aspects environment, application scenarios, requirements, functions, active structure, behavior, system of objectives and shape. The principle solution is concretized during the conceptual design [22]. To describe the different aspects the specification technique **CONSENS** (CONceptual design Specification technique for ENgineering of complex Systems, cf. Sect. 4.1) is used. The principle solution creates a common understanding of the development task and the system between the developers of the different domains involved. Based on the principle solution the developer can design and develop the system in the involved domains of mechanical, software, control and electrical/electronic engineering.

The reference process for the domain-spanning conceptual design consists of four main phases "Planning and Clarifying the Task", "Conceptual Design on the System Level", "Conceptual Design on the Subsystem Level" and "Concept Integration" (cf. Fig. 3.3). In "**Planning and Clarifying the Task**" the design task of the system and the requirements are identified and evaluated. The results are the list of requirements, the environment model, the recommended product structure type and its design rules, as well as the application scenarios. Based on the previously determined requirements on the system, solution variants are developed in the "**Conceptual Design on the System Level**". These solution variants are evaluated within the phase. Based on the results, the best one will be chosen and consolidated into the principle solution on the system level. This includes the identification of potential for the use of self-optimization, based on contradictions within the principle solution for the system that can be solved either by compromise or by self-optimization. The principle solution for the self-optimizing system on the system level is the result of this phase. Based on this, the system is modularized and

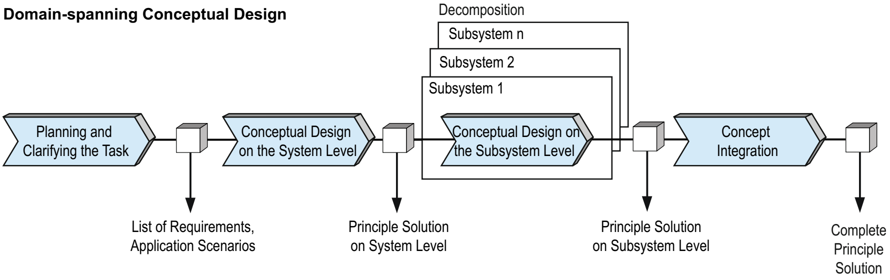

Domain-spanning Conceptual Design
Decomposition
Subsystem n
Subsystem 2
Subsystem 1
Planning and Clarifying the Task
Conceptual Design on the System Level
Conceptual Design on the Subsystem Level
Concept Integration
List of Requirements, Application Scenarios
Principle Solution on System Level
Principle Solution on Subsystem Level
Complete Principle Solution

**Fig. 3.3** The four main phases of the domain-spanning conceptual design

a principle solution for each single subsystem is developed in the "**Conceptual Design on the Subsystem Level**". This procedure corresponds to the conceptual design on the system level, starting out with planning and clarification of the task. The result of this phase is presented by the principle solutions on the subsystem level. This process can be conducted recursively, because the subsystem itself can be a system with subsystems and so forth. The principle solutions for the subsystems are integrated into one principle solution, which represents the complete system, within the phase "**Concept Integration**". Afterwards the principle solution is analyzed regarding its economic efficiency, dynamical behavior and dependability. In this analysis phase contradictions between the principle solutions on the subsystem level are identified. Again it will be checked, if these contradictions can be solved by self-optimization. The result of this phase is the principle solution for the complete system that serves as the starting point for the subsequent domain-specific design and development. This is carried out in parallel in the specific domains (mechanical engineering, electrical/electronic engineering, control engineering and software engineering) [23]. Section 3.3 describes this process in detail.

In "**Planning and Clarifying the Task**" (Fig.3.4), the development task is abstracted in the first step ("Analyze the Task") and its core is identified. This is followed by an analysis of the environment which investigates the most important boundary conditions and influences on the system ("Analyze the Environment"). The external objectives (e.g. "maximize comfort") emerge as well as disturbances ("Identifiy External Objectives"). Beyond that, consistent combinations of influences which are called situations are formed. By the combination of characteristic situations with system states, application scenarios occur that describe a part of the whole functionality of the system, which has to be developed ("Identify Application Scenarios"). By using the structuring procedure by Steffen (2007) it is possible to identify an adequate product structure type for the system (cf. Sect. 4.6). The results of the first phase are documented by demands and requests within the list of requirements ("Identify Requirements") [23].

Based on the list of requirements the main functions of the system are identified and set into a function hierarchy in the "**Conceptual Design on the System Level**" (Fig. 3.5). Each function has to be fulfilled to satisfy the requirements ("Define the Function Hierarchy"). Therefore solution patterns are sought, which can execute the

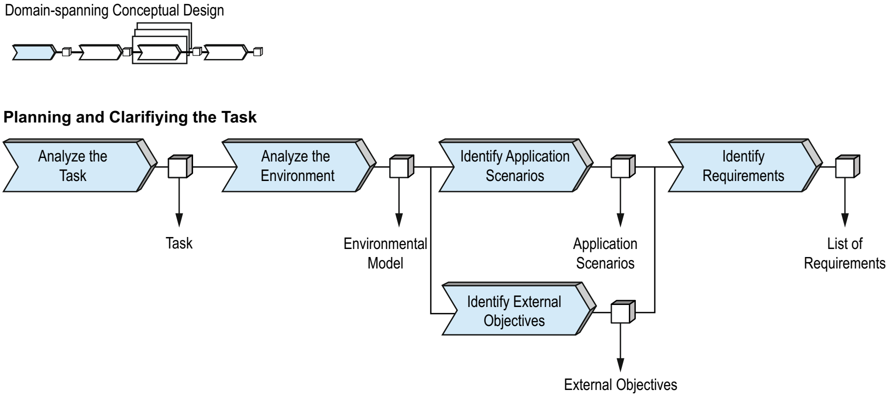


**Fig. 3.4** Planning and Clarifiying the Task

desired functions ("Identify Solution Patterns"). Solution patterns represent reusable expertise for problem-specific solutions in generic mode (cf. Sect. 4.5). Within a morphologic box the solution patterns are combined to consistent solution variants ("Develop Solution Variants"). A consistency analysis is used, in order to determine useful combinations of solution patterns of the morphologic box [38]. In the next step, the solution variants are evaluated and the most promising chosen ("Evaluate and Choose Solution Variants"). In this step more than one solution variant can be promising, the developers have to decide individually, whether they concretize one or more promising solution variants. Then the selection of one solution variant can be done based on more detailed information at a future date in the development process. The resulting solution must not be self-optimizing at this stage. This is only the case, if self-optimizing functions are explicitly requested in the list of requirements. Otherwise a mechatronic solution variant results. The potential for the use of self-optimization is identified later in this phase.

The consistent bundle of solution patterns form the basis for the active structure. The active structure describes the characteristics of the system elements as well as their cross-linking ("Define Active Structure"). Based on the active structure, an initial construction structure can be developed, because there are primal details on the shape within the system elements ("Define Shape"). In addition, the systems behavior is modeled in the step "Define System Behavior". In this step the application scenarios are formalized and the respective behavior is analyzed regarding its consistency (cf. Sect. 4.3). Basically, this concerns the activities, states and state transitions of the system as well as the communication and cooperation with other systems and subsystems. Subsequently the principle solution is analyzed regarding conflicting objectives. A potential for self-optimization is given, if the changing influences on the system requires modifications of the pursued objectives. The system needs to adjust its behavior ("Identify Conflicting Objectives"). In this case the system of objectives is developed ("Develop System of Objectives", cf. Sect. 4.4) and the list of requirements extended ("Extend the Reqirements"). Based on the new

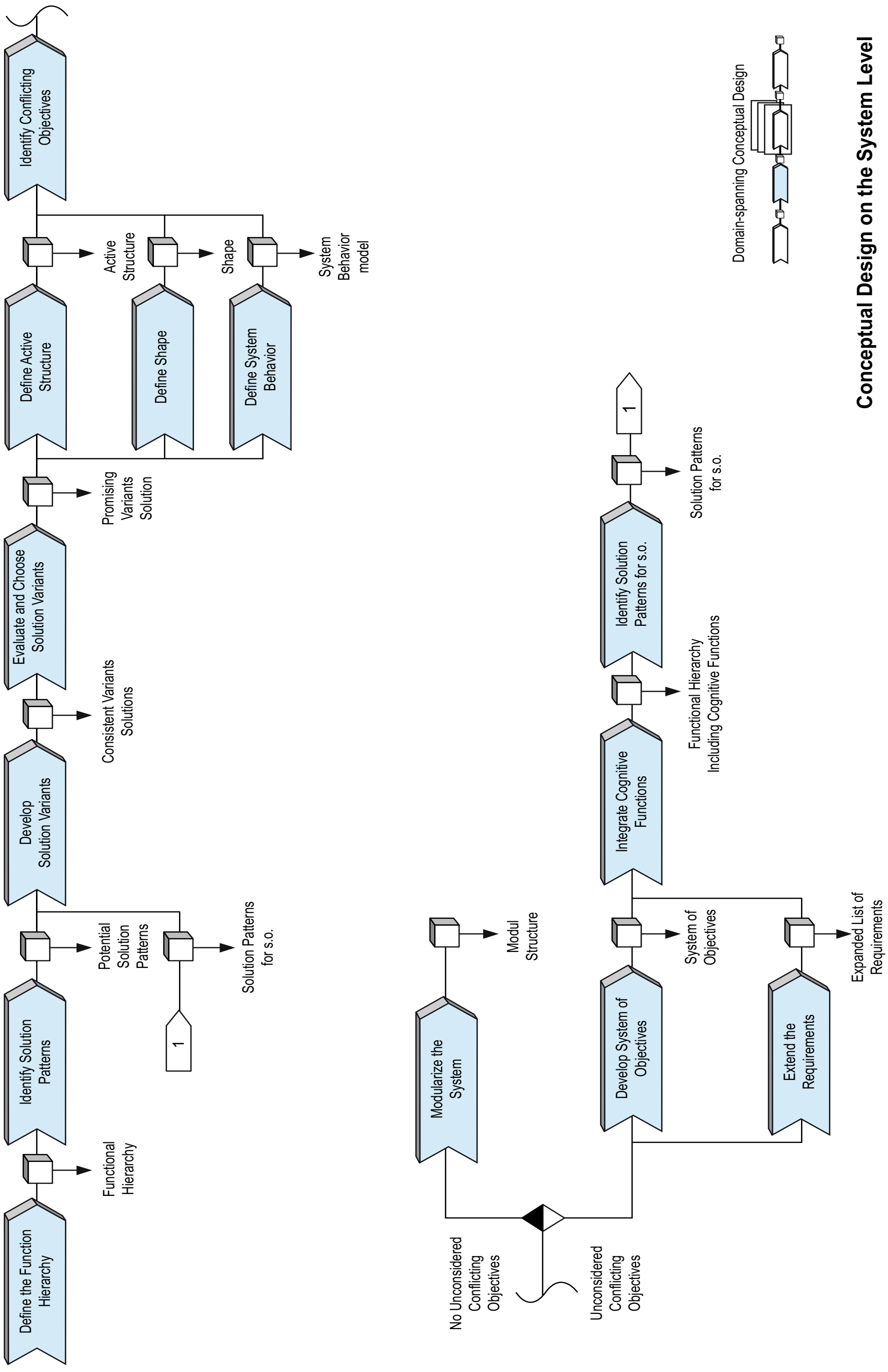

Conceptual Design on the System Level
Domain-spanning Conceptual Design
Unconsidered Conflicting Objectives
No Unconsidered Conflicting Objectives
Modularize the System
Modul Structure
Develop System of Objectives
System of Objectives
Extend the Requirements
Expanded List of Requirements
Integrate Cognitive Functions
Functional Hierarchy Including Cognitive Functions
Identify Solution Patterns for s.o.
Solution Patterns for s.o.
1
Define the Function Hierarchy
Functional Hierarchy
Identify Solution Patterns
Potential Solution Patterns
Solution Patterns for s.o.
1
Develop Solution Variants
Consistent Variants Solutions
Evaluate and Choose Solution Variants
Promising Variants Solution
Define Active Structure
Active Structure
Define Shape
Shape
Define System Behavior
System Behavior model
Identify Conflicting Objectives

**Fig. 3.5** Conceptual Design on the System Level

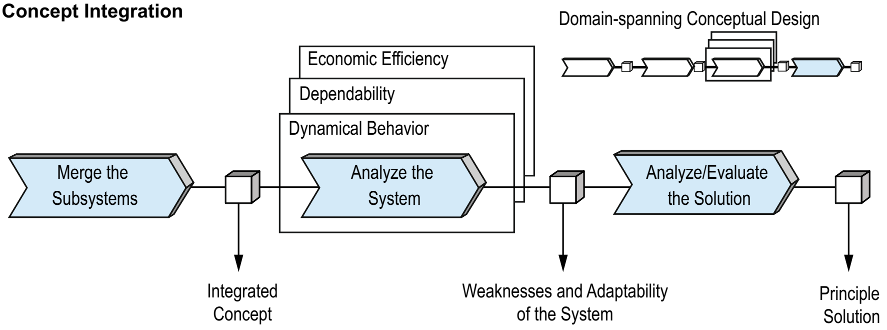


**Fig. 3.6** Concept Integration

requirements, cognitive functions are chosen and the function hierarchy complemented. The tasks that need to be done in the step "Integrate Cognitive Functions" are explained in Sect. 4.5. For these functions solution patterns for self-optimization are identified to enable self-optimizing behavior continuously ("Identify Solution Pattern for Self-Optimization"). Generally a pattern describes a recurring problem in our environment and the core of the solution to that [6]. Especially in the context of self-optimizing systems the pattern approach is an established instrument to externalize and store the knowledge of experts. The overall objective is to integrate specialized knowledge for self-optimizing algorithm during the conceptual design (cf Sect. 4.5 and 5.3.12). The resulting changes and extensions of the system structure and the system behavior need to be included appropriately for the selected solution pattern (Link 1). In preparing the self-optimization concept, the self-optimization processes will eventually be defined, the absence of conflicts of the self-optimization process will be analyzed and the conditions, in which the self-optimization has to be working in, will be defined as well. This iteration (see Fig. 3.5) runs as long as all conflicting objectives are taken into account. If all conflicting objectives of the system are considered the system is modularized into subsystems ("Modularize the System").

In the next phase **"Conceptual Design on the Subsystem Level"** the principle solution for the subsystems are developed. This is done in the same way as in the conceptual design on the system level. The conceptual design is an iterative process, because subsystems are systems as well and can therefore be modularized to subsystems, too.

To make sure that the specified subsystems are not in conflict, the phase **"Concept Integration"** is performed on each system hierarchy level (Fig. 3.6). In the first step "Merge the Subsystems", the principle solutions on the subsystem level are merged to a detailed principle solution on the system level. The final concept is analyzed regarding the dynamical behavior, dependability and economic efficiency before the design and development is initiated ("Analyze the System").

**Dynamical Behavior:** For the analysis of the dynamical behavior of the system an idealized simulation model is built based on the aspects active structure and behavior-states [8]. To make sure, that the controlled basic structure of the system can realize the specified dynamical behavior, the resulting requirements and constraints for the mechanical structure are identified. Based on these the plant model of the simulation model can be expanded by idealized actuators and sensors to design a first control concept. This is secured by the idealized model. The analysis results of the secured control concept help to detail the aspects active structure and behavior-states [16].

**Dependability:** Due to the autonomous adaptation of the system behavior of a self-optimizing system during operation, the dependability of the system is of high priority. It has to be secured over the entire development process. In the conceptual design specially developed methods can be used, such as the combined Failure Mode and Effect Analysis (FMEA) and the Fault Tree Analysis (FTA) based on the principle solution (cf. Sect. 4.7) [14]. To support the developers with search, selection and planning of dependability engineering methods, which are suitable for their particular development task, a methodology has been developed. Further information about the methodology and some dependability methods are given in [24, D.o.S.O.M.S. Chap. 2]

**Economic Efficiency:** Besides the technical feasibility, economical aspects also contribute to the decision to further design and develop a self-optimizing system, because it can result in changing resource requirements in the development, production and operation when compared to a conventional mechatronic solution. For this reason the economic efficiency of the developed solution needs to be proven during the conceptual design. Section 4.8 presents a method to identify the costs and benefits of a self-optimizing product concept to prove its economic efficiency.

In the last step "Evaluate the Solution", the solution is evaluated and the decision is made, if the system, will be designed and developed further in the following domain-specific design and development.

## 3.3 Domain-Specific Design and Development

Sebastian Korf, Mario Porrmann, Katharina Stahl, Oliver Sudmann, and Mareen Vaßholz

The principle solution is the starting point for the domain-specific design and development. It contains the information that builds the basis for the domain-specific development tasks. The transition from the principle solution to the domains involved is described briefly in Sect. 5.1. In classical mechatronic development processes, the four domains mechanical, control, software, and electrical/electronic engineering are involved. To realize the self-optimization process for the system optimization needs to be performed. For this task the expertise of higher mathematics and artificial intelligence is needed. The tasks of the domains are executed in parallel, where possible. Therefore, the consistency of the system models needs to be ensured, by continuously integrating the results of the domains in the (sub)system integration.

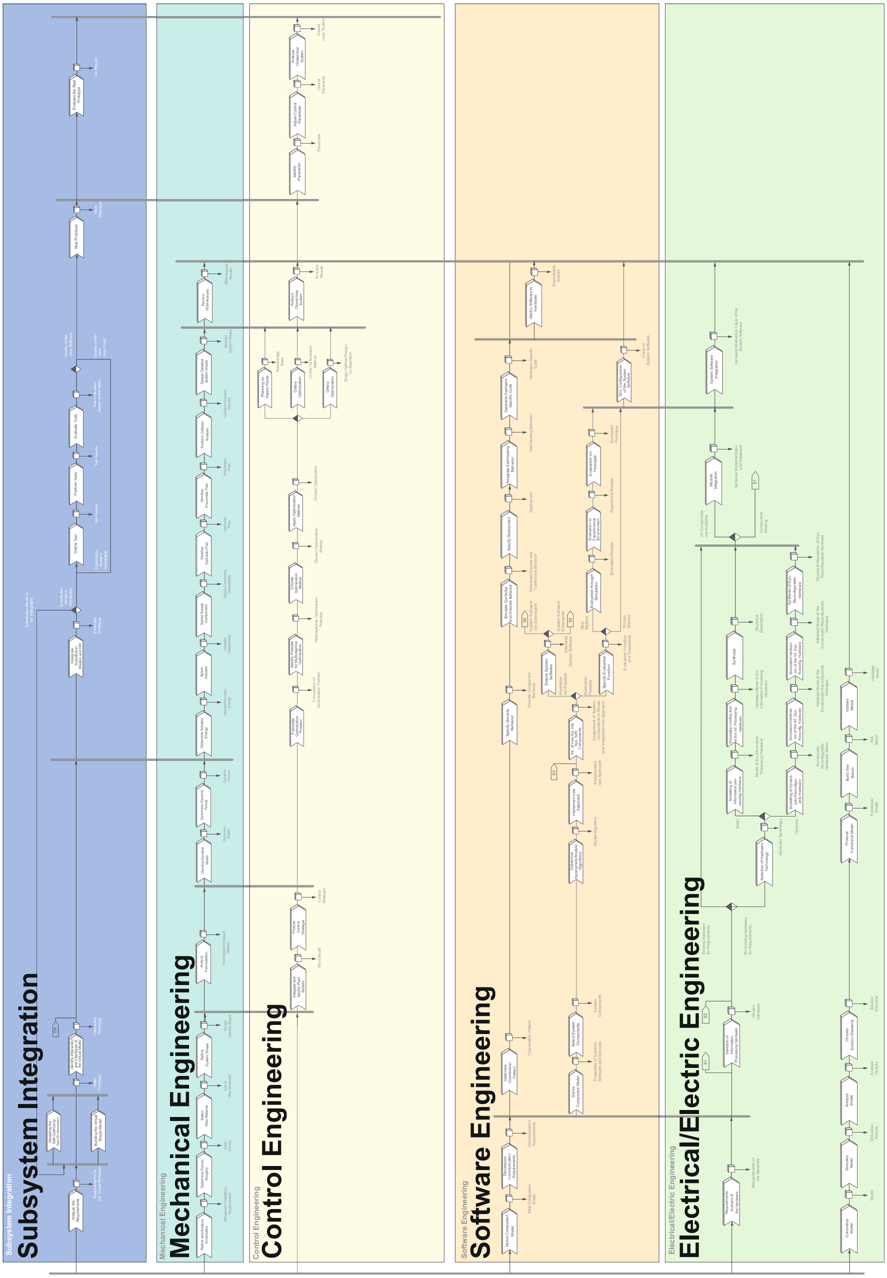


**Fig. 3.7** Schematic representation of the domain-specific design and development

This allows a short response time to failures and therefore reduces time and cost intensive iterations.

Fig. 3.7 gives an overview of the process for one subsystem. As modeled in the conceptual design, the system itself consists of subsystems. Therefore, the presented process for the design and development is recursive and conducted at different hierarchy levels of the system.

The presented **reference process** for the domain-specific design and development of self-optimizing systems shows an ideal approach. It is based on our experiences from the development of the RailCab and its function modules. Depending on the development project, the system type and the environment of the development project this process needs to be implemented for the specific development task.

The domains work in parallel where possible to reduce development time and costs. In the domain mechanical engineering the mechanical basic structure with actuators and sensors of the self-optimizing system is developed (cf. Sect. 3.3.1). To guarantee the desired dynamical behavior of the system the domain control engineering designs the controller for the system (cf. Sect. 3.3.2). In control engineering the optimization strategy for the system is developed for the Cognitive Operator as well. The results of the optimization phase are implemented into the control strategy. The domain software engineering develops the system software and the discrete software to realize the reconfiguration of the hardware and the communication between different system components (cf. Sect. 3.3.3). The reconfigurable hardware is designed and developed within the electronic engineering to realize the high flexibility that is needed to enable the autonomous change of the systems behavior. Furthermore, self-optimizing systems demand high power transmission. The power electronic is developed by electrical engineering (cf. Sect. 3.3.4). The results of the domains are integrated continuously in an overall system model. Their interaction is tested as a virtual prototype, to identify failures at an early stage (cf. Sect. 3.3.5). Important synchronization points are depicted within the process, where the domains exchange their results and get information that are needed for the further development. In Sect. 3.3.6 the interaction of the domains in the design and development is explained.

Even though the process gives the impression that it is stringent, especially at these synchronization points iterations can emerge. But they are also possible at every stage of the process. The amount and order of the iterations depends on the design objects, organizational boundary conditions and the developers individual behavior as well as on the specific use of methods. Therefore, the application of the presented approach for the design and development to a specific development task and company has to be tailored individually (cf. Sect. 3.4).

### *3.3.1 Mechanical Engineering*

The aim of the domain mechanical engineering is the design and development of the **shape model** for the self-optimizing system (cf. Fig. 3.8). The starting point is the principle solution. On its basis the kinematics of the multibody system are refined and analyzed ("Refine and Analyze Kinematics"). The result is a list of movement possibilities and displacements of the system. The kinematic model of

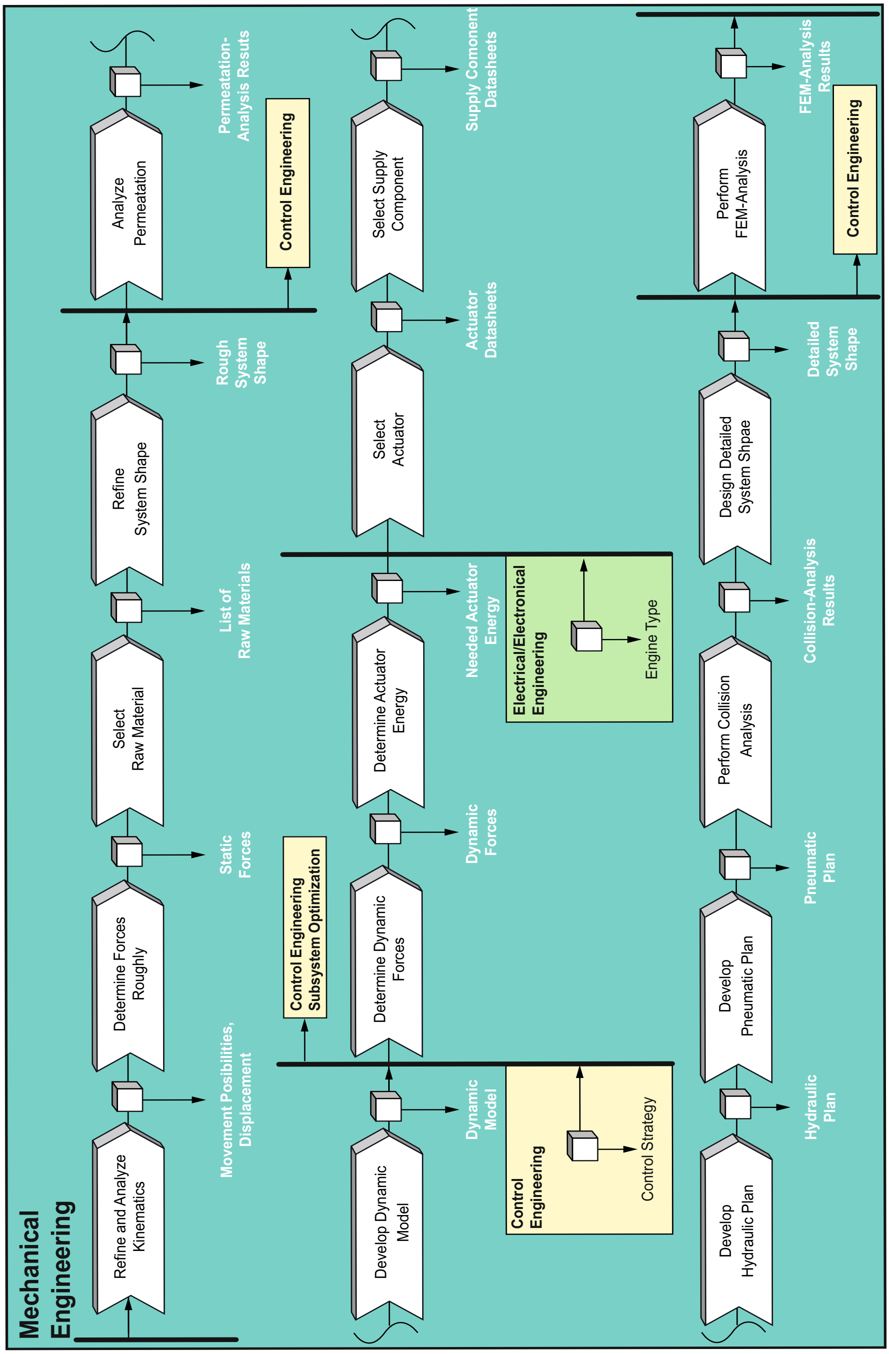


**Fig. 3.8** Development process for the domain mechanical engineering

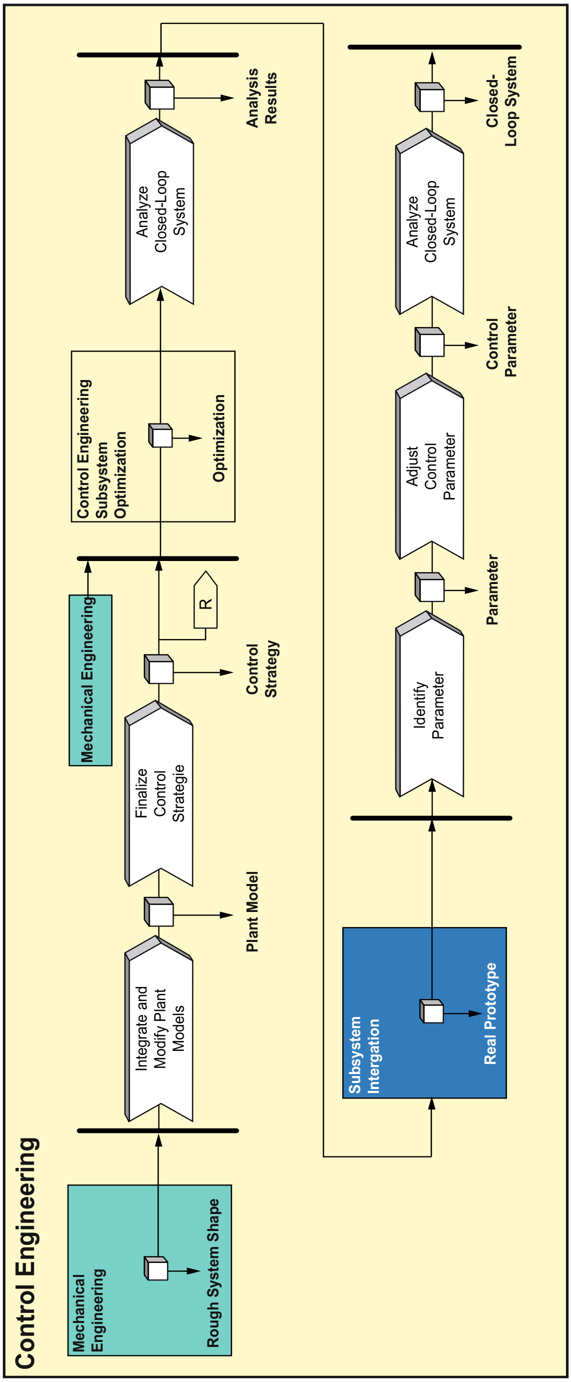
the system allows the determination of the forces that affect the system, like tensile or compressive forces. An output object of the step "Determine Forces Roughly" are the static forces. Now that the systems kinematic and static forces are known, the appropriate raw materials can be selected that meets the resulting requirements ("Select Raw Material"). In the next step "Refine System Shape" the aspect shape of the principle solution is detailed in the 3D-CAD software tool. Based on this rough system shape model, the permeation of the bodies are analyzed ("Analyze Permeation"). If the analysis shows no permeation conflicts, the results of the domain control and electrical engineering are integrated into the dynamic model in the step "Develop Dynamic Model". Furthermore, the shape model serves as input for the "System Integration", where the first virtual prototype can be implemented. Based on the dynamic model the dynamic forces can be determined ("Determine Dynamic Forces") and control engineering can design the optimization strategy for the self-optimizing system parallelly. Now the domain mechanical engineering can determine the needed actuator energy for the system ("Determine Actuator Energy") and select an actuator that meets the requirements ("Select Actuator"). The chosen engine type by electrical engineering is integrated as well. In the next step "Select Supply Component" the supply components are selected. Then the hydraulic and pneumatic plan for the system are developed ("Develop Hydraulic Plan"; "Develop Pneumatic Plan"). Afterwards a collision analysis is performed ("Perform Collision Analysis") and in case, that the analysis results are satisfactory, the shape of the system is designed in detail ("Design Detailed System Shape") and integrated into the overall system model [35].

### *3.3.2 Control Engineering*

The domain control engineering designs the controller to guarantee the desired dynamical behavior of the self-optimizing system (cf. Fig. 3.9. The main results are the components in the Operator-Controller-Module (OCM, cf. Sect. 1.3). However, particular parts of the two operator levels can also be defined by control engineering. To realize the control strategy the shape model of the system serves as input. Already in the domain-spanning conceptual design a simplified controller is designed as part of the principle solution (cf. Sect. 3.2). By integration of the shape model and the corresponding parameters from the principle solution, the plant model can be modified in the step "Integrate and Modify Plant Model". Based on this, the control strategy can be finalized. Feedforward and feedback controllers as well as observers are designed in this step. This step represents the main task of control engineering. A multitude of different techniques to perform this task are known and this step can deviate sharply for different systems. Additionally, this step can be quite complex. Thus, the process needs to be tailored for a specific controller design task. After the design of the controller, the dynamic model can be integrated, which is the basis for the system optimization (cf. Fig. 3.10.

The dynamic model, integrates the results of the domains mechanical, electrical and control engineering, that can be simulated by appropriate tools (e.g. Matlab

**Fig. 3.9** Development process for the domain control engineering



Simulink[7] to name only one state of the art simulation tool). In the first step "Identify Potential for Multiobjective Optimization" an analysis of the dynamic model, returns system properties which are important for simulation (e.g. integration step sizes appropriate for the time scale of the system's behavior) and especially for optimization (e.g. which parameters are design variables, of what type are they and how big is their influence on the system's behavior). Aims and restrictions on the (sub)system have been listed before in the conceptual design. Now they have to be formally specified into a vector of mathematical objectives (e.g. a formula measuring the system's energy or the passengers' comfort) and constraints (e.g. a constraint envelope on the system's states or box constraints on the controls). If there are more than one single objective and if they are contradictory, i.e. a better value of one objective leads to a worse achievement of the other objectives, there is potential for multiobjective optimization. Simulations with different sets of parameters can help to identify contradictory objectives and in the end, the multiobjective optimization procedure itself precisely reveals the set of optimal compromises between different objectives that can be achieved (cf. Sect. 1.4.1). The mathematical objectives and constraints along with the dynamical model of the system define the (multiobjective) optimization problem ("Formulate Optimization Problem"). Depending on its nature, an appropriate optimization method has to be chosen ("Choose Optimization Method", cf. Sect. 1.4.1 and Sect. 5.3 for examples of subclasses and appropriate methods). These methods can be behavior-based and/or model-based. In case that an **behavior-based optimization method** is chosen, the planing domain needs to be defined ("Define Planning Domain").

The planning domain can be the RailCab for example. In the next step the planing model is formulated ("Formulate Planning Model"). This model is an abstraction of the reality and includes for the RailCab e.g. track sections. For this planning model Pareto fonts need to be generated using **model-based optimization methods**. In both cases the chosen optimization method is applied to generate an initial set of (Pareto) optimal configurations of the design parameters ("Apply Optimization Method").

The optimization of the system during runtime can proceeded in three different ways, each one requiring a different approach during the development.

1) **"Planning on Pareto Points":** To allow self-optimization of the module, a planning algorithm can be chosen that selects current design configurations from a knowledge base during runtime, i.e. a finite set of Pareto optima during runtime (cf. the hybrid planning method described in Sect.5.3.8).

2) **"Online Optimization":** Alternatively, a model based optimization method can be implemented on the system that allows for an online optimization (e.g. of a scalarized multiobjective optimization problem by path following methods) during operation.

3) **"Offline Optimization":** From the set of Pareto optimal points, one specific optimum is selected and the corresponding design configuration is fixed or made constant for the remaining development process. In case of multiobjective optimiza-

[7] For further information see `http://www.mathworks.de/`

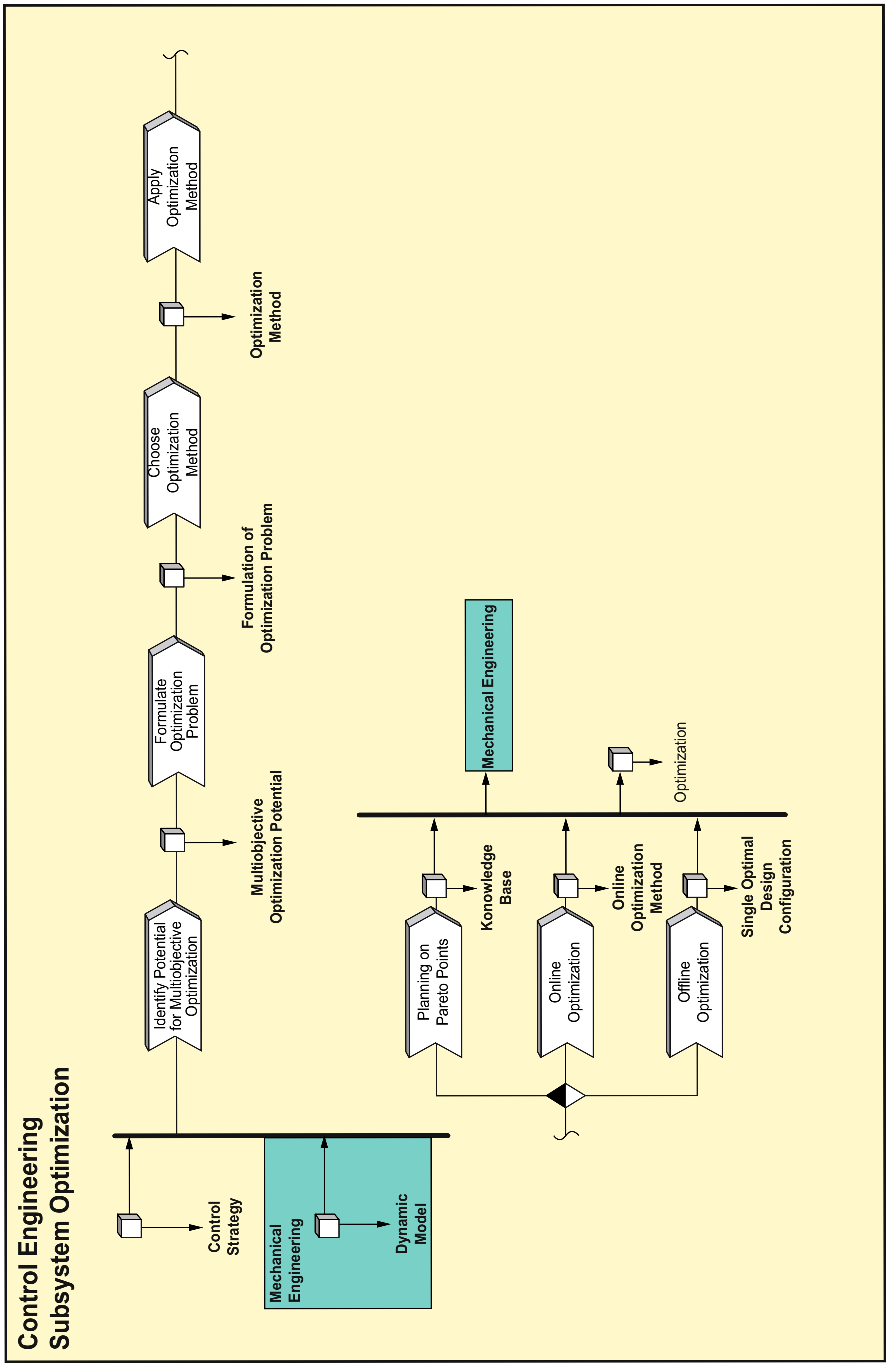

Control Engineering
Subsystem Optimization
Control Strategy
Mechanical Engineering
Dynamic Model
Identify Potential for Multiobjective Optimization
Multiobjective Optimization Potential
Formulate Optimization Problem
Formulation of Optimization Problem
Choose Optimization Method
Optimization Method
Apply Optimization Method
Planning on Pareto Points
Konowledge Base
Online Optimization
Online Optimization Method
Offline Optimization
Single Optimal Design Configuration
Mechanical Engineering
Optimization

**Fig. 3.10** Development process for the (sub)system optimization

tion this means to choose and fix once and for all a weighting (prioritization) of the different objectives.

Variants 2) and 3) can be also combined such that at runtime, precomputed parameters from a knowledge base are used until updated parameters have been computed. A typical example of optimization parameters are the control parameters, e.g. for feedback control laws. The planning algorithms are integrated into the Cognitive Operator.

The results of the system optimization are integrated and the closed-loop system is simulated and analyzed according to its dynamic behavior ("Analyze Closed-Loop System"). A first version of the closed-loop system and characteristic values, like switching times are handed over to the domain software engineering. Furthermore, it is integrated into the virtual prototype. After testing the control strategy in the virtual prototype a real prototype for chosen parts of the system is built and tested as hardware-in-the-loop. The measured data needs to be compared with the simulation results to identify uncertain parameters of the real system ("Identify Parameter"). For this parameter identification, a multitude of methods are given in literature. These uncertainties may result from missing data concerning the components used and especially the interaction of the different components. Examples for those parameters are moments of inertia. Depending on the identified system parameters an adjustment of the control parameters can be necessary to guarantee the desired operational behavior ("Adjust Control Parameter").

After this adjustment the controller is finalized and the characteristic values, like switching times are available for further development of the system. In order to test the desired behavior the adapted system with the finalized parameters is simulated ("Adjust Closed-Loop System"). The finalized closed-loop system is given back to the domain of software engineering. These three steps may be necessary again, after building the real prototype of the whole system, when the comparison of the measured data and the simulation results show deviations.

### 3.3.3 Software Development

The domain software engineering focuses on the development of the discrete software and the system software. The discrete software coordinates the interaction among the components and its reconfigurations. In particular, discrete software triggers controller reconfigurations to adapt the system's behavior or changes the software structure to different communication needs. Of course, this requires synchronizations of the domains control and software engineering, i.e. control engineers hand-over interface specifications of the controllers, message time intervals for the communication, and the switching durations of the controller reconfigurations [31].

These different software configurations demand varying resources. The system software must consider the resource demands on reconfigurable hardware. Accordingly, synchronizations with the domain electrical engineering are necessary, i.e. the system software must consider properties of the dynamically reconfiguring hardware. In the following we will first describe the design and development process of the discrete software. Afterwards we will introduce the system software design process.

We apply the MECHATRONICUML method [9, 17, 27] to develop the **discrete software**. The key concepts of MECHATRONICUML are a component-based system model which enables scalable compositional verification of safety-properties, the model-driven specification and verification of reconfiguration operations [17, 30, 42], and the integration of the discrete software with the controllers of the mechatronic system [12]. Section 5.2 introduces MECHATRONICUML in detail. Figure 3.11 provides an overview of the MECHATRONICUML design and development process and its integration with the domain control engineering and the system software design process [28, 31].

In "Derive Component Model", a software architect uses the active structure (Sect. 4.1) and the application scenarios (Sect. 4.3) to derive a set of components describing the structure of the system. A component is a software entity that encapsulates a part of the system behavior which implements a certain functionality. Each component defines a set of external interaction points for accessing its functionality. An initial component model can be derived with a semi-automatic model transformation approach (Sect. 5.1.2.1) [25].

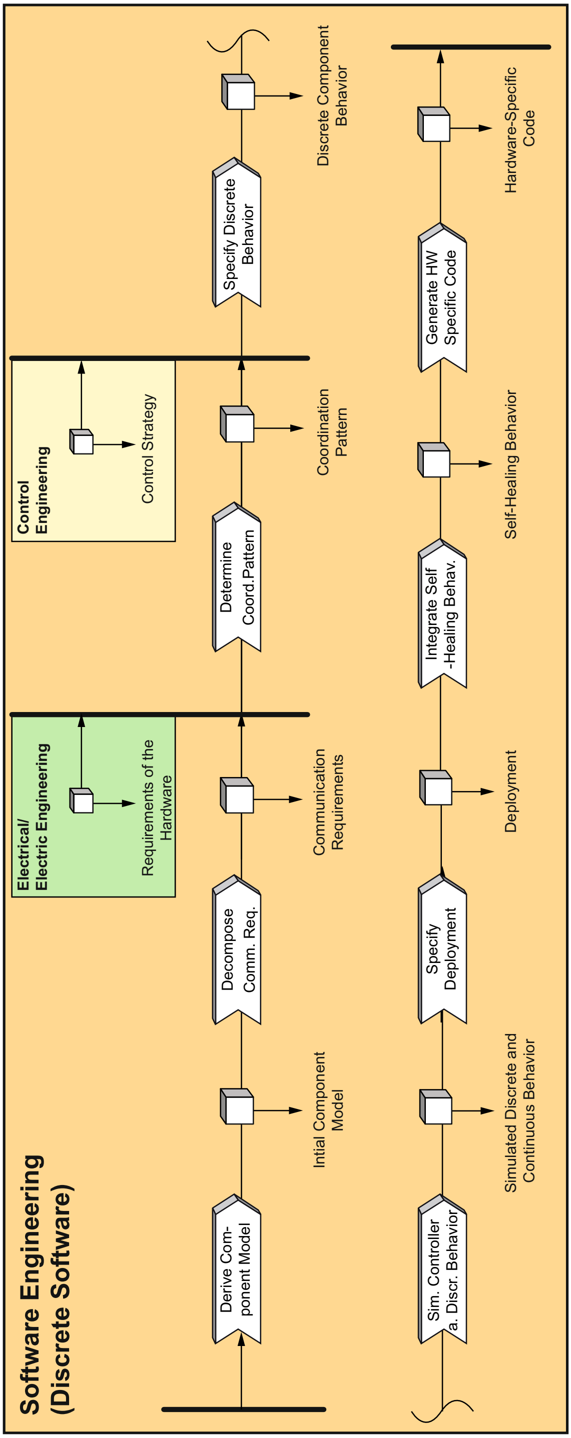
Components do not work in isolation, but they collaborate for realizing the intended system functionality. A system engineer describes the collaboration with Modal Sequence Diagrams (MSDs), a formal variant of UML sequence diagrams [2], for all Application Scenarios during the conceptual design. As the conceptual design addresses all involved domains, the communciation behavior specification is often only a coarse-grained description that a software engineer must refine in the next steps. Of course, the refinement results in a more complex behavior that is hard to analyze efficiently. Therefore, a software architect decomposes the specified MSDs into smaller, reusable sets of MSDs in "Decompose Communication Requirements". These smaller sets of MSDs form the communication requirements for the system software and the next steps of MECHATRONICUML.

In "Determine Coordination Pattern", a software architect determines real-time coordination patterns, that describe communication protocols by real-time statecharts, a combination of UML state machines [2] and timed automata [7, 11]. Based on the communication requirements, real-time coordination patterns can either be reused, or new real-time coordination patterns can be specified and verified.

Afterwards, component engineers specify the component's communication behavior based on the real-time coordination patterns and integrate the controllers to specify the reconfiguration behavior ("Specify Discrete Behavior"). Of course, this requires a synchronization with the domain control engineering, i.e. control engineers must hand over the interface specification of the controllers, and the duration to change from one controller configuration to another. We call these durations switching times.

The component behavior is safety-critical. In particular, engineers must ensure that the reconfiguration behavior does not include harmful configurations, and that all hard real-time constraints can be met. Component engineers use formal methods such as model checking to guarantee these safety-properties for the discrete behavior.

**Fig. 3.11** Development process for the discrete software of the domain software engineering


Software Engineering (Discrete Software)
Derive Component Model
Intial Component Model
Decompose Comm. Req.
Communication Requirements
Electrical/ Electric Engineering
Requirements of the Hardware
Determine Coord.Pattern
Coordination Pattern
Control Engineering
Control Strategy
Specify Discrete Behavior
Discrete Component Behavior
Sim. Controller a. Discr. Behavior
Simulated Discrete and Continuous Behavior
Specify Deployment
Deployment
Integrate Self -Healing Behav.
Self-Healing Behavior
Generate HW Specific Code
Hardware-Specific Code

In order to fulfill the optimization objectives, the system must select an appropriate reconfiguration and still ensure all safety-properties. Component engineers integrate a planning technique, called safe planning (cf. Sect. 5.2.4.5), to ensure that the system selects only reasonable reconfigurations.

Formal methods do not scale well for the analysis of hybrid behavior, i.e. the combination of discrete components and continuous controller behavior. Instead in the step "Simulate Controller and Discrete Behavior", software and control engineers collaborate to simulate the complete hybrid system. Software engineers can apply model transformation techniques to integrate the discrete with the continuous controller behavior [29].

Next, electrical engineers hand over the specification of the hardware platform. A deployment engineer uses this information to specify the mapping of all software components to hardware nodes in the step "Specify Deployment".

Although formal methods and simulation can prevent systematic flaws of the behavior specification, random failures of the hardware are still a risk for safety-critical tasks. In the step "Integrate Self-healing Behavior", a hazard analyst identifies dangerous combinations of failures [26] and extends the reconfiguration behavior to enable self-healing of the system (Sect. 5.2.7).

At this point, the models of the discrete components and the controllers are translated to hardware-specific code and integrated with the system software. In fact, system software engineers must provide the configuration of the system software, i.e. the interface specification, such that software engineers can perform an appropriate code generation.

The **system software** is the system layer interconnecting the reconfigurable hardware and the software layer. The process to develop self-optimizing system software is a strongly coorperative approach that requires interaction between the respective domains. Figure 3.12 provides an overview of the system software's development process and its interaction with other domains. Design decisions within a related domain may affect the system software layer and raise changes in the system software design. Hence, continuous cooperation and coordination during the entire design and development process is essential between all the related domains.

Self-optimization is a run-time property of a system which is often realized by rule-based (and often threshold-based) autonomous decision making. It is therefore important to ensure that the system's self-optimizing methods are able to satisfy the given objectives at operation time. This leads to an iterative process for the development of self-optimizing system software that is characterized by three main phases:

1. Identification and Conception
2. Development and Implementation
3. Evaluation

As a prerequisite, we assume the self-optimizing system software to be composed of reusable components that are activated in respect to the present system requirements. Each system component addresses a specific system function or property. In the context of real-time operating systems, a component might be represented by a kernel function or an operating system service. This approach provides the basis for reconfiguration within the system software. An architecture for such a self-optimizing real-time operating system is described in Sect. 5.5.

Considering the design and development process of system software we are starting with **Identification and Conception** and the step "Derive Requirements on System Software". Here, the basic requirements on the system software can directly be derived from the principle solution. Additionally, the requirements of the software and the hardware layer flow together and determine the detailed requirements and properties of the system software. As each component is assigned to a specific system property, in the step "Select System Components", we identify the components or types of components that contribute to realize the defined system properties. Recurring properties will be addressed by providing a pool of already implemented components. These components do not need to be implemented from scratch any more. Reusing these existing components only requires an integration and configuration by performing "SCL Configuration of System Software".

Development, Implementation and Evaluation phases are omitted when taking advantage of reusable components. However, these two phases become essential for the development of a novel component which is related to a system requirement that has not been raised before.

In the **Development and Implementation** phase, the appropriate models or algorithms to address the requirements, e.g. concerning objective of the self-optimizing method, have to be determined in "Determine Appropriate Models/Algorithms" before implementing the new approach ("Implement New Approach"). Then, the new component requires the integration into the system software and the existing system components by "Integrate New Approach into System Software and System Software Components". This integration might incorporate an adaptation or adjustment of the existing system components in the step "Extend System Software".

In some cases, requirements on the extension of the system software might be contradictorily and inconsistent with the existent design which will make a simple extension impossible. Such a case could not necessarily be foreseen at a former development phase. An adjustment of the developed new approach will be performed in "Integrate New Approach into System Software and System Software Components". In other cases, the integration of a specific component may require further components that have to be developed and we have to start again from the development step "Determine Appropriate Models/Algorithms".

The aim of the **Evaluation** phase is to verify whether the new system component operates towards the desired objective. Because of the nature of self-optimization, it does not always need to be obvious at design or development phase. If the resulting system behavior does not match the system objectives, adjustment either in the concept of the method, its implementation and integration, or the evaluation functions are required. This process has to be iteratively conducted until the developed self-optimizing method meets its requirements.

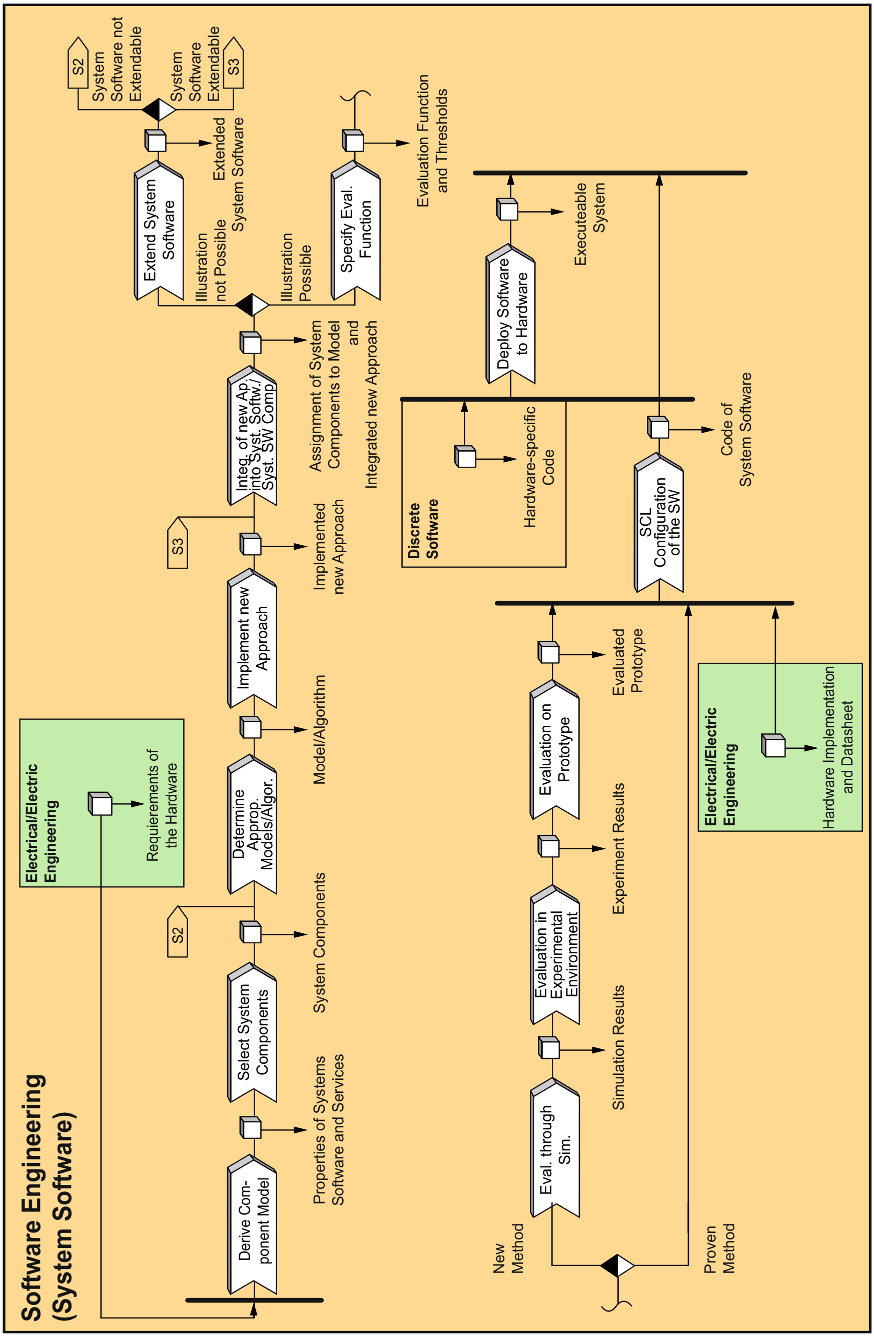

Software Engineering
(System Software)
Electrical/Electric Engineering
Requierements of the Hardware
Derive Component Model
Properties of Systems Software and Services
Select System Components
System Components
S2
Determine Approp. Models/Algor.
Model/Algorithm
Implement new Approach
Implemented new Approach
S3
Integ. of new Ap. into Syst. Softw./ Syst. SW Comp.
Assignment of System Components to Model and Integrated new Approach
Illustration not Possible
Illustration Possible
Extend System Software
Extended System Software
System Software not Extendable
S2
System Software Extendable
S3
Specify Eval. Function
Evaluation Function and Thresholds
New Method
Proven Method
Eval. through Sim.
Simulation Results
Evaluation in Experimental Environment
Experiment Results
Evaluation on Prototype
Evaluated Prototype
Electrical/Electric Engineering
Hardware Implementation and Datasheet
SCL Configuration of the SW
Code of System Software
Discrete Software
Hardware-specific Code
Deploy Software to Hardware
Executeable System

**Fig. 3.12** Development process for the system software of the domain software engineering

### *3.3.4 Electrical and Electronic Engineering*

Self-optimizing systems demand high flexibility, to be able to adapt the behavior during operation. Dynamically reconfigurable hardware can ensure this flexibility. On the other hand they demand high power transmission. Therefore, in the domain electrical/electronic engineering both microelectronic devices for the information processing and power electronics are developed.

The design of new **microelectronic devices** for information processing in self-optimizing systems comes with an increasing complexity compared to todays mechatronic systems, and a high demand for flexibility, which can be achieved, e.g. by utilizing dynamically reconfigurable hardware. In order to allow an automatic adaptation of the hardware to dynamically changing environments, the well-established design-flows for microelectronic systems need to be extended. Figure 3.13 depicts the process to create statically as well as dynamically reconfigurable microelectronic hardware for self-optimizing systems. Starting from the principle solution the requirements of the hardware are analyzed ("Requirement Analysis of the Hardware"). Based on these requirements, a set of appropriate information processing hardware components is selected ("Selection of Information Processing Hardware"). If all the requirements can not be fulfilled with existing hardware, new components need to be implemented. In this case, as a first step, the hardware technology (e.g. embedded processor architecture, FPGA technology, semiconductor technology) is chosen ("Selection of Hardware Technology"). In the next step of the process it is defined, if the new hardware will contain only static components or static and dynamic ones. In the first case, a standard design flow is used for the hardware development:

1) The information processing hardware needs to be modeled ("Modeling of Information Processing Hardware"). In accordance to the classical Y-diagram [21], the hardware can be specified on different abstraction levels (e.g. circuit level, logic level, register-transfer level, algorithmic level, or system level) in three different domains (behavior, structure, and geometry). In the domains of structure and behavior, a description of the hardware in a Hardware Description Language (HDL) is prevalent.

2) A simulation and verification step of the new hardware model ensures the correct implementation of the hardware ("Simulation/Verification of the Information Processing Hardware").

3) In the synthesis step, the hardware model is transformed into a structural description in the chosen technology ("Synthesis").

If dynamic reconfiguration of the hardware is foreseen, the design flow has to be extended with respect to the special requirements of these architectures. Therefore, in a first step the best suited technique for dynamic reconfiguration has to be selected ("Selection of Methods for Dynamic Reconfiguration"). Depending on the granularity of the reconfigurable architecture, different variants of the subsequent process steps are executed (cf. Sect. 5.4.1 and Sect. 5.4.2).

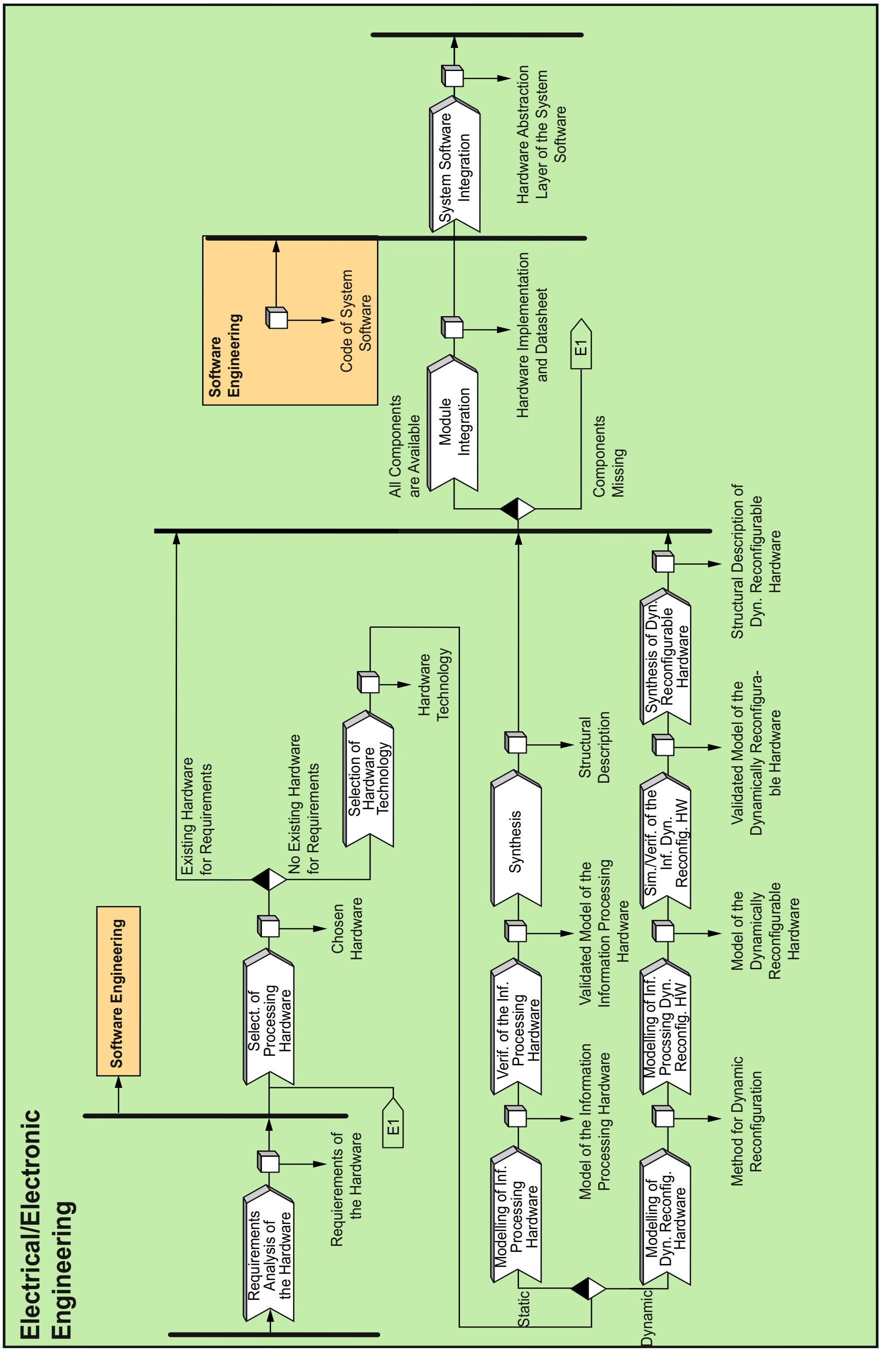

Electrical/Electronic Engineering
Requirements Analysis of the Hardware
Requierements of the Hardware
Software Engineering
Select. of Processing Hardware
E1
Chosen Hardware
Existing Hardware for Requirements
No Existing Hardware for Requirements
Selection of Hardware Technology
Hardware Technology
Static
Dynamic
Modelling of Inf. Processing Hardware
Model of the Information Processing Hardware
Verif. of the Inf. Processing Hardware
Validated Model of the Information Processing Hardware
Synthesis
Structural Description
Modelling of Dyn. Reconfig. Hardware
Method for Dynamic Reconfiguration
Modelling of Inf. Procssing Dyn. Reconfig. HW
Model of the Dynamically Reconfigurable Hardware
Sim./Verif. of the Inf. Dyn. Reconfig. HW
Validated Model of the Dynamically Reconfigura-ble Hardware
Synthesis of Dyn. Reconfigurable Hardware
Structural Description of Dyn. Reconfigurable Hardware
All Components are Available
Components Missing
E1
Module Integration
Hardware Implementation and Datasheet
Software Engineering
Code of System Software
System Software Integration
Hardware Abstraction Layer of the System Software

**Fig. 3.13** Development process for the microelectronic devices in the domain electrical/electronic engineering

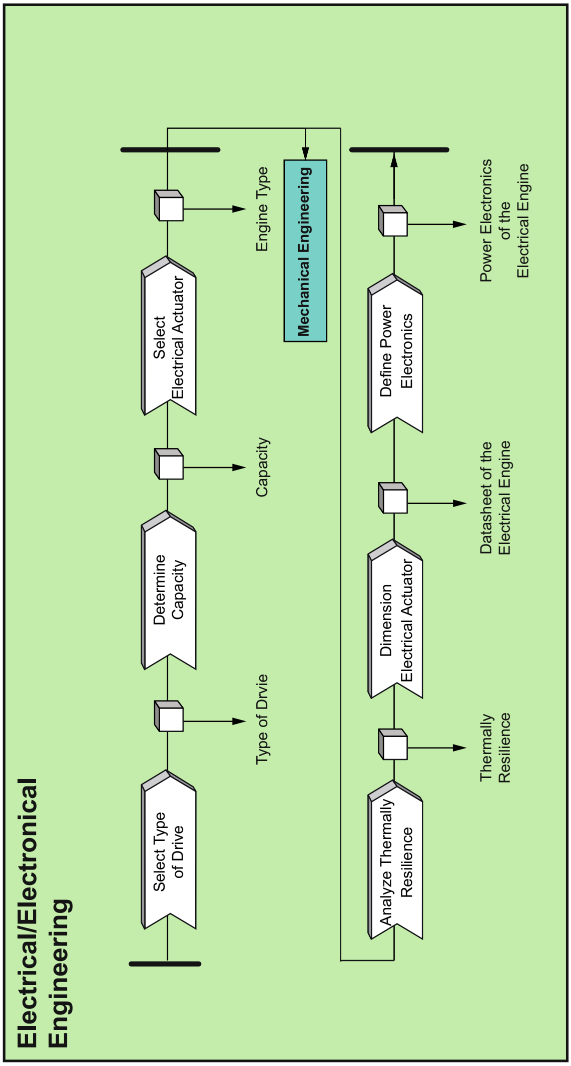

Electrical/Electronical Engineering
Select Type of Drive
Type of Drvie
Determine Capacity
Capacity
Select Electrical Actuator
Engine Type
Mechanical Engineering
Analyze Thermally Resilience
Thermally Resilience
Dimension Electrical Actuator
Datasheet of the Electrical Engine
Define Power Electronics
Power Electronics of the Electrical Engine

**Fig. 3.14** Development process for the power electronics in the domain electrical/electronic engineering

The complete design flow for FPGA-based dynamic reconfiguration has been integrated within the design environment INDRA (Integrated Design Flow for Reconfigurable Architectures, cf. Sect. 5.4.4.1). For the modeling ("Simulation/Verification of the Information Processing Dynamically Reconfigurable Hardware") of a complex dynamic reconfigurable system, the PALMERA layer model has been implemented (cf. Sect. 5.4.3.1). The simulation ("Simulation/Verification of the Dynamically Reconfigurable Hardware") is supported by a Hardware-in-the-Loop design environment, specifically targeting dynamically reconfigurable systems (cf. Sect. 5.4.4.3). The synthesis step ("Synthesis of Dynamically Reconfigurable Hardware") is an integral part of the INDRA environment. The steps "Selection of

Hardware Technology" to "Synthesis of Dynamically Reconfigurable Hardware" are repeated until all hardware requirements are fulfilled. Finally, all hardware components are integrated in a complete module ("Module Integration") and the system software is ported to the new architecture ("System Software Integration"). Examples of platform applications with dynamic reconfiguration are shown for the minirobot BeBot in Sect. 5.4.5.2 and for Multiprocessor architectures in Sect. 5.4.5.3.

The necessary steps for the design and development of the **power electronics** are presented in Fig. 3.14. The aspects requirements and active structure serve as input for this task. Based on these, the type of electrical drive is selected that fulfills the requirements ("Select Type of Drive"). For the drive the capacity is determined to ensure that the demanded consumer load of the system components can be realized ("Determine Capacity"). In the next step "Select Actuator" the appropriate engine is chosen. The information about the engine are handed over to the mechanical engineering domain and integrated into the system model. Afterwards the thermal resilience of the engine is tested ("Analyze Thermally Resilience"). Then the data sheet of the electrical engine is derived ("Dimension Electrical Actuator") and the power electronics can be defined ("Define Power Electronics"). The result of the task is integrated into the overall system [35].

### *3.3.5 (Sub)system Integration*

The domains work in parallel in the design and development of a self-optimizing system. In the (sub)system integration their results are integrated continuously into the **overall system model** ("Integrate Domain Results"). The synchronization of the overall system model with the domain specific models is supported by the model synchronization technique described in Sect. 5.1. In case that the changes in the overall system model by one domain are relevant for other ones, these changes are transfered. Thus the consistency of the different models is ensured over the entire development process. To analyze the interaction of the different parts, a virtual prototype is built. Based on the principle solution a requirement analysis for virtual prototyping is performed ("Analyze the Requirements"). The principle solution gives evidence for the **virtual prototype**, but in the ongoing development process, new requirements can occur and need to be considered, too. With these requirements we will be able to build the virtual shape model ("Build the Virtual Shape Model") in parallel with the modeling of the **self-optimizing test environment** ("Model the Self-Optimizing Test Bench"), in which the virtual prototype will be tested. After completing these two steps we get a static prototype of our system and its environment. To be able to test various aspects of the virtual prototype, we have to identify interaction and interfaces ("Identify Interaction and Interfaces of Virtual Model"). This is needed when a specific part of the virtual prototype is to be tested later on. When, e.g. we want to test a spring damper system, we have to identify the actors in the virtual model and define an interaction and an interface for it. With this we have the ability to place inputs in and receive results out of the system under test.

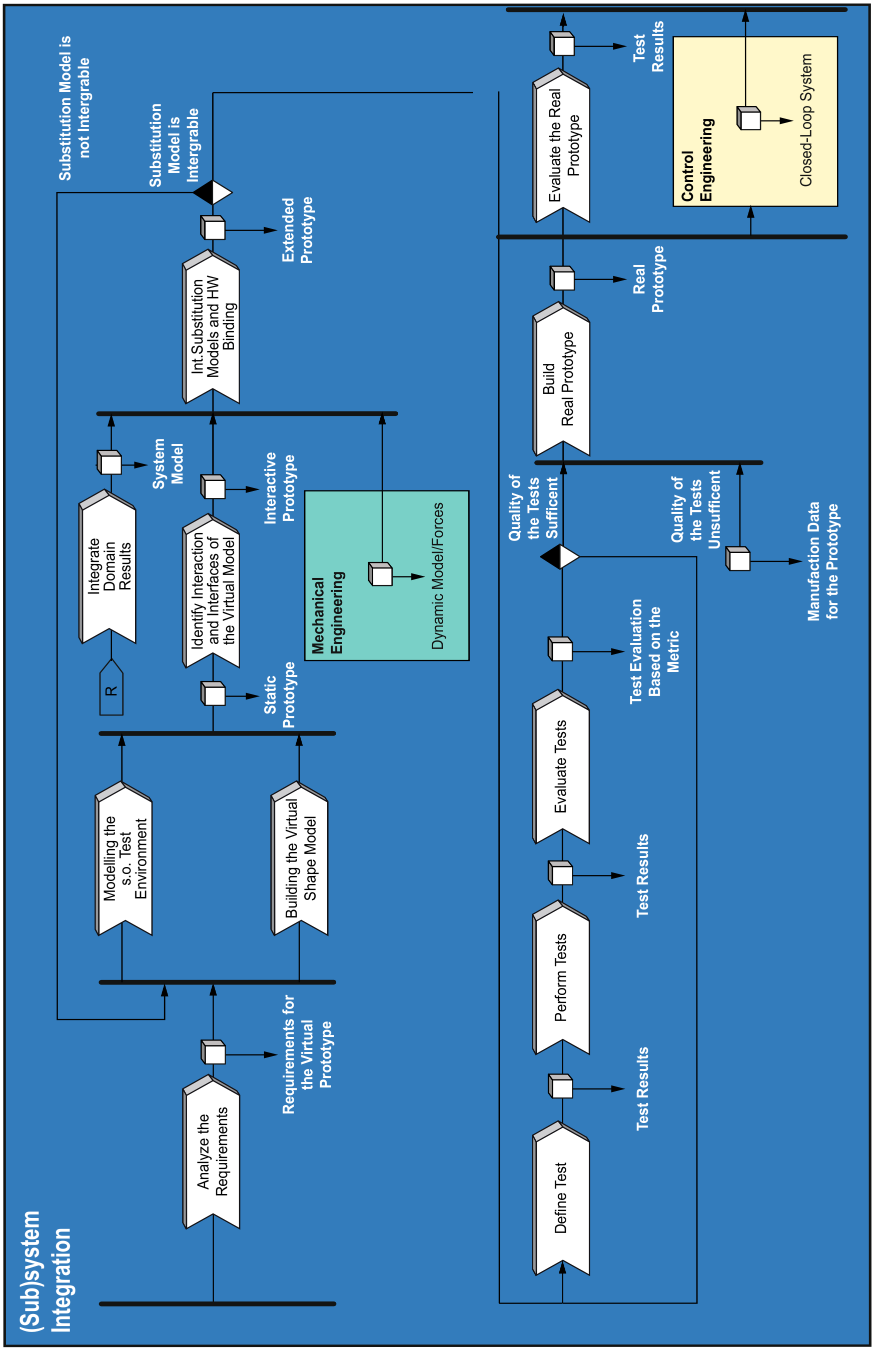

(Sub)system Integration
Analyze the Requirements
Requirements for the Virtual Prototype
Modelling the s.o. Test Environment
Building the Virtual Shape Model
Static Prototype
R
Integrate Domain Results
System Model
Identify Interaction and Interfaces of the Virtual Model
Interactive Prototype
Mechanical Engineering
Dynamic Model/Forces
Int.Substitution Models and HW Binding
Extended Prototype
Substitution Model is Intergrable
Substitution Model is not Intergrable
Define Test
Test Results
Perform Tests
Test Results
Evaluate Tests
Test Evaluation Based on the Metric
Quality of the Tests Sufficent
Quality of the Tests Unsufficent
Manufaction Data for the Prototype
Build Real Prototype
Real Prototype
Evaluate the Real Prototype
Test Results
Control Engineering
Closed-Loop System

**Fig. 3.15** Integration and test of the results of the different domains in the (sub)system integration

When the interaction and interfaces are identified we get an interactive virtual prototype. This interactive prototype is ready for testing by the other domains. When a domain wants to do a test series with the virtual prototype they have to integrate the substitution models and/or do the hardware binding ("Integrate Substitution Models and Hardware Binding"). For example when control engineering wants to test a PID controller as part of the system, they may integrate a substitution model of this controller into the virtual prototype (mostly done in earlier iterations when only the model of the controller exists) or, in later phases, bind the real hardware to the virtual prototype. In case the virtual prototype is not exact enough to be able to do a test accurately enough, we have to refine the self-optimizing test environment as well as the virtual shape model and identify the interaction and interfaces of the improved virtual prototype in the steps "Model the Self-optimizing Test Environment" to "Integrate Substitution Models and Hardware Binding". After the substitution model is integrated the tests can be defined ("Define Test"). After that step we get test cases and an evaluation metric. Now the tests can be performed on the virtual prototype ("Perform Test"). The test results can be evaluated and we get a test evaluation based on the metric we defined earlier ("Evaluate Test"). To be sure, that the tests are significant we check the quality. If the quality is not sufficient enough we do another iteration, defining new tests, performing them and evaluating the results (cf. Sect. 5.6).

When all subsystems in all domains are implemented and built, we are able to build the real prototype ("Build Real Prototype") and evaluate it ("Evaluate Real Prototype"). After the successful test of the prototype the production documents can be finalized and the production of the system is initiated.

### *3.3.6 Interaction of the Domains in the Design and Development*

In the previous sections the tasks of the domains involved as well as the integration of their results were described briefly. To successfully realize a self-optimizing system the performance of the different tasks and also their interactions are essential. Important synchronization points are depicted within the reference process, where the domains exchange their results and get information that are needed for the further development. Fig. 3.16 illustrates the interaction of the domains in the design and development from an **object oriented development** point of view.

Based on the principle solution the domains start with their specific development tasks. The domain mechanical engineering starts with the design of the system shape. By the time a rough shape model of the basic structure for the system is developed, the model is handed over to the domain control engineering (synchronization point (1)). The shape model is integrated into the plant model of Control engineering and the control strategy can be designed. This strategy defines the dynamic behavior of the system. Therefore, it needs to be integrated into the dynamic model of the domain mechanical engineering. Furthermore, the control strategy is an important input for the domain software engineering, to define the communication between the system components (synchronization point (2)). In parallel, the domain software engineering has identified the communication requirements for the system components and the domain electrical/electronic engineering has identified the requirements for the dynamically reconfigurable hardware, based on the principle solution. The identified requirements are exchanged at synchronization point (3) between the two domains. Based on the requirements the design of the system software can be initiated. Meanwhile the (sub)system integration has developed an interactive prototype for the virtual test of the system and has integrated the results of the domains continuously into the overall system model. At synchronization point (4) the dynamic model of the system is handed over to the (sub)system integration, where it is merged with the interactive prototype. The dynamic model is also the basis for the development of the optimization strategy in the control engineering. In parallel to the development of the optimization strategy, the system shape is detailed by the domain mechanical engineering. The shape model is extended by actuators and sensors, as well as by the engine type for the system, that has been selected by the experts from electrical engineering. In the next step in electrical engineering the power electronics of the engine are selected. The detailed system shape is passed over to control engineering, where it is integrated into the closed-loop system together with the optimization strategy at synchronization point (5). The closed-loop system is simulated afterwards. In the interim the hardware was developed by the domain electrical/electronic engineering. At synchronization point (6) the hardware implementation and the corresponding data sheet are exchanged with the domain software engineering, where the code for the system software is derived. In parallel the hardware abstraction layer of the system software and the hardware specific code are developed. The three design objects are integrated at synchronization point (7). The software is ported to the hardware resulting in an executable system.

Now the design objects of the domains involved are integrated into the overall system model at synchronization point (8) and the real prototype of the sub(system) is build (synchronization point (9)). Based on the tests of the prototype the domain control engineering ensures, that the simulated parameters do not differ from the test results. In that case, the control strategy needs to be adjusted. After synchronization point (10) and the successful test of the prototype the manufacturing documents can be finalized and the production of the system is initiated.

The presented reference process is recommendatory for the development of self-optimizing systems, for a specific development task further development steps can be necessary. Therefore it needs to be implemented to a development project individually, the resulting process sequence provides the framework for the development. Nevertheless, during execution changes in the development objectives can result that make adjustments necessary. To support the project manager with changes in the planned procedure, we adopted the concept of self-optimization to the management of the development, which will be explained in the following section.

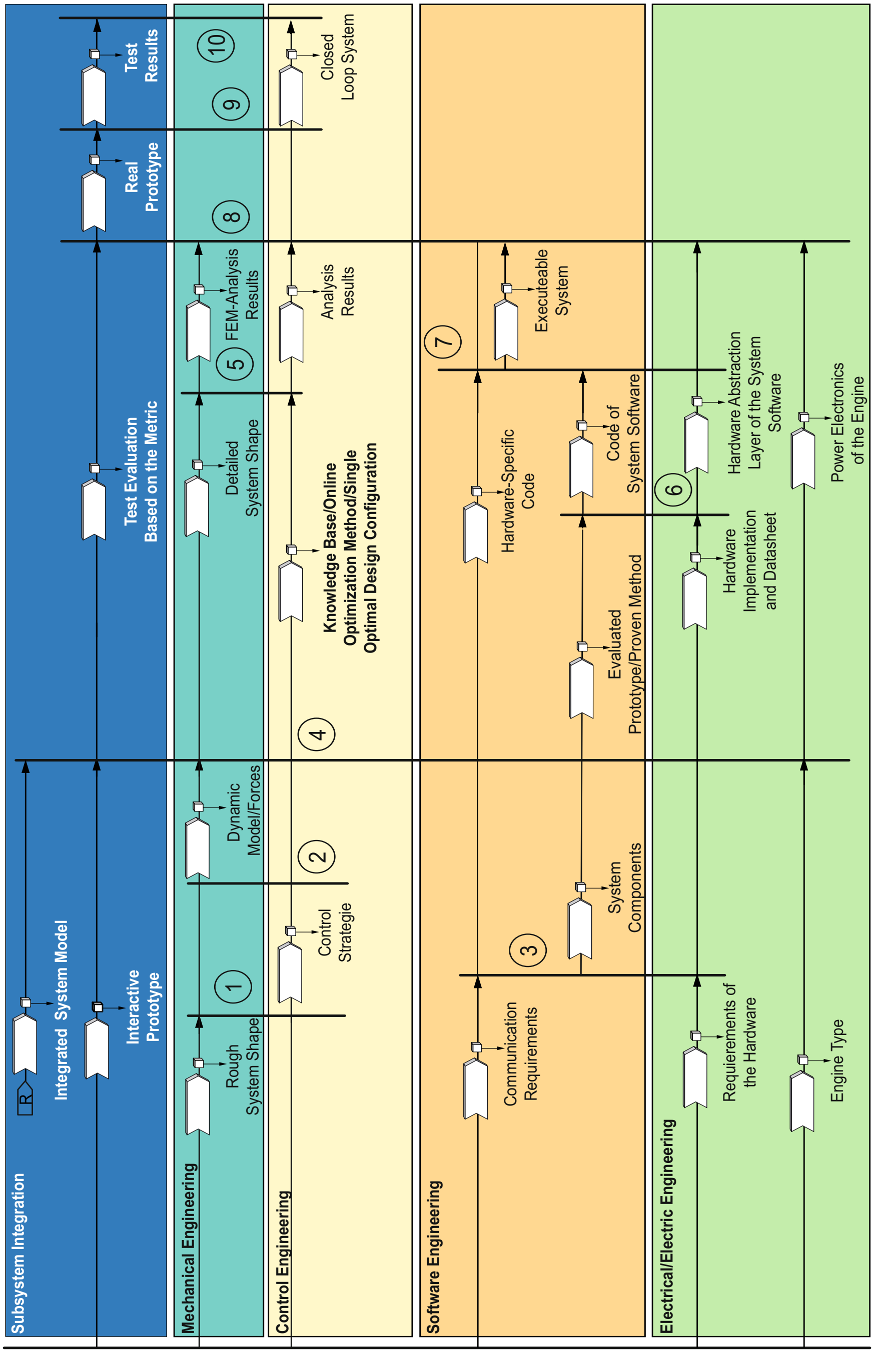


**Fig. 3.16** Interaction of the domains at the synchronization points

## 3.4 Self-optimizing Development Process

Tobias Gaukstern and Oliver Sudmann

Development processes for self-optimizing systems are characterized by an increasing communication and coordination effort. The reasons for this are the increasing interdisciplinary nature as well as the increasing system and interface complexity. Hence, the process is inherently complex.

As a consequence of the interdisciplinary nature, the development process must consider several synchronizations between the domains. Most synchronizations are project-specific. Project managers add them according to the system structure and the overlapping domain-specific models.

Furthermore, development projects are often long-term projects. During a long-term project, strategic goals can change. In addition, changes to the system often occur. Therefore, project managers must adapt the project to these changes during a project many times.

As a starting point, project managers can implement the introduced reference process for their specific development task. In particular, the process of the ”Design and Development” phase must consider characteristics of the developed system and the strategic goals of the organization.

As development processes can consist of thousands of process steps, the tailoring of a process is time-consuming and error-prone. Therefore, project managers need methods to **plan** complex and dynamic development processes. In the following, we present a combination of two approaches to address these challenges. First, we present a **framework for a self-optimizing development process** by Kahl (2013) (Sect. 3.4.1). The framework adapts the paradigm of self-optimization for the planning of the development process. Based on the analysis of the actual development situation, the weighting of the strategic goals and thus the development process are adjusted. The result is a situation-specific development process that fits with the strategic goals and the system under development.

The framework adds known synchronizations by means of process rules to the development process. Usually, an organization is not aware of all possible synchronizations. We, therefore, apply our second approach to plan synchronizations systematically (Sect. 3.4.2). The approach refines the process further and adds synchronizations between the domains according to the interdependencies of the models systematically.

### *3.4.1 Framework of a Self-optimizing Development Process*

During the development of self-optimizing systems, unexpected, dynamic and changeable situations occur. These situations cannot be controlled by the available process management methods. The obvious solution is to transfer mechanisms to increase the performance of mechatronic systems in managing the development processes. Therefore we will introduce the framework by Kahl (2013) that depicts the transformation of the paradigm of self-optimization on development processes and provides appropriate procedures for its application [35].

#### 3.4.1.1 Aspects of the Self-optimizing Development Process

The systematic model of a self-optimizing development process consists of a process management and development activities (Fig. 3.17). The **self-optimizing process management** adapts the development activities according to all relations affecting the development process, which are denoted as influences. The sum of all influences results in the development situation at a specific point in time.

The development activities are connected through relations between their input- and output-objects. Its entirety describes the object of development. According to the involved disciplines the development objects can be divided into discipline-spanning and discipline-specific objects. The development objects are generated through resources, which can be divided in personnel resources and impersonal resources (machines, methods, software tools).

The development objects are transformed through development activities, according to the system of objectives, which is determined by the process management. The system of objectives describes the desired, claimed, or avoided behavior of the development process. They are classified in external, inherent and internal objectives: External objectives are set from the outside of the development process, e.g. objectives regarding the time limit. Further, they result from properties of the development object. Inherent objectives reflect the purpose of the development process, e.g. "minimize number of staff turnover". Internal objectives are objectives, which are relevant at a specific point in time. Their weighting is adapted to development situations through the self-optimizing development process.

The entirety of development activities, action and development objects and their relations result in the structure of the development process. Their properties are the parameter of the process. Together they form its behavior. The behavior depicts the execution of the development activities and the resulting transformation of input- into output-objects at a specific point in time. According to the resulted remarks the defined aspects of self-optimization can be transferred to the development process of advanced mechatronic systems:

*”We understand a development process’ self-optimization as an endogenous adaptation of the objectives of the development process on changing operating conditions, as well as a resulting target-oriented adaptation of the parameters and, if necessary, of the structure and therefore the behavior of the development process”.*

#### 3.4.1.2 Sequence of the Self-optimizing Process

The self-optimization of the development process takes place as a process that consists of the following three actions (cf. Sect. 1.2):

1. **Analyzing the current situation:** The current situation of the development process is gathered by using monitoring variables (e.g. progress of the development objects, elapsed development time) as well as the influences, affecting the

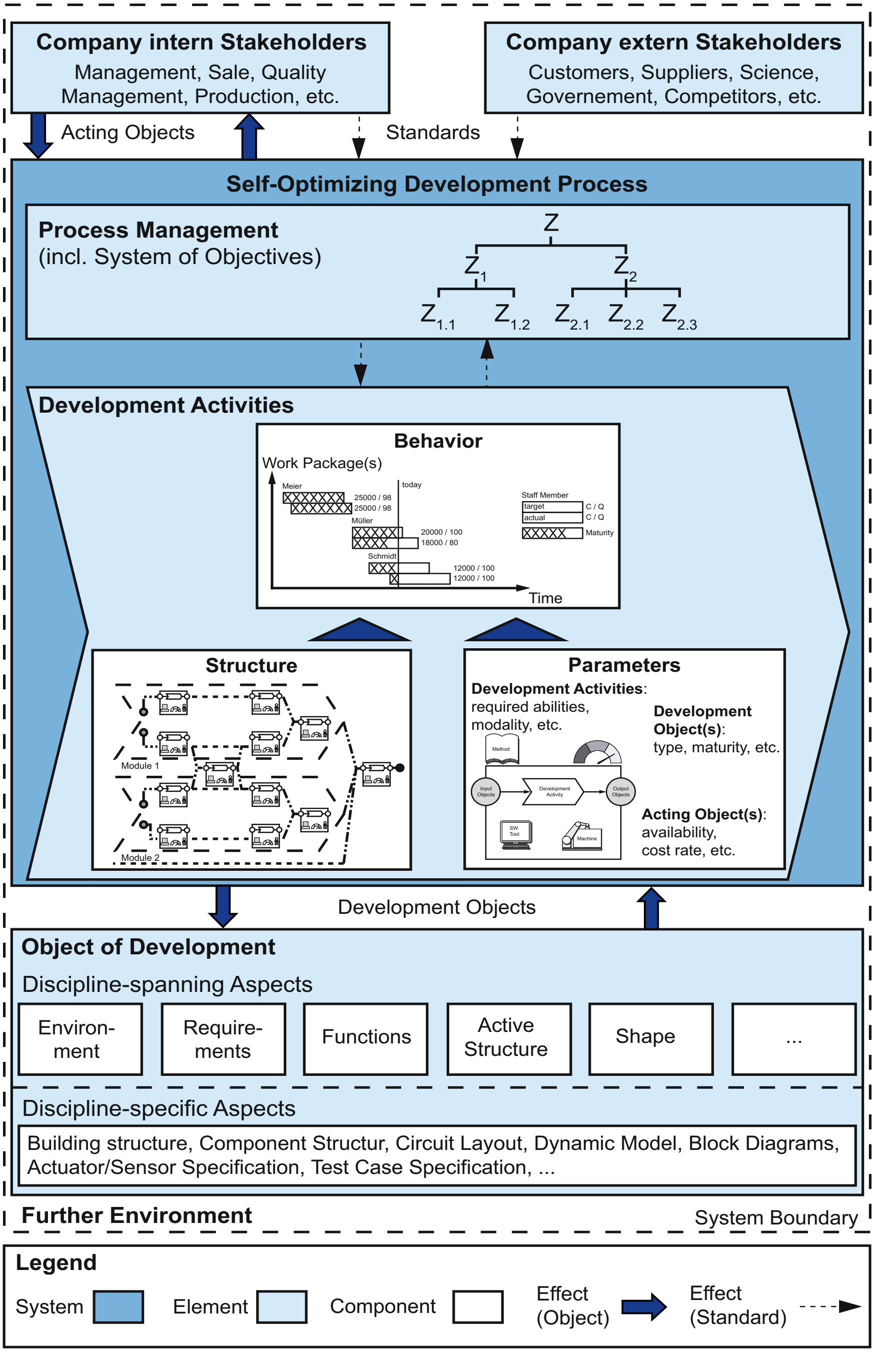

Company intern Stakeholders
Management, Sale, Quality Management, Production, etc.
Company extern Stakeholders
Customers, Suppliers, Science, Governement, Competitors, etc.
Acting Objects
Standards
Self-Optimizing Development Process
Process Management
(incl. System of Objectives)
Z
Z1
Z2
Z1.1
Z1.2
Z2.1
Z2.2
Z2.3
Development Activities
Behavior
Work Package(s)
Meier
today
25000 / 98
25000 / 98
Müller
20000 / 100
18000 / 80
Schmidt
12000 / 100
12000 / 100
Staff Member
target
actual
C / Q
C / Q
Maturity
Time
Structure
Module 1
Module 2
Parameters
Development Activities: required abilities, modality, etc.
Development Object(s): type, maturity, etc.
Method
Input Objects
Development Activity
Output Objects
SW Tool
Machine
Acting Object(s): availability, cost rate, etc.
Development Objects
Object of Development
Discipline-spanning Aspects
Environ-ment
Require-ments
Functions
Active Structure
Shape
...
Discipline-specific Aspects
Building structure, Component Structur, Circuit Layout, Dynamic Model, Block Diagrams, Actuator/Sensor Specification, Test Case Specification, ...
Further Environment
System Boundary
Legend
System
Element
Component
Effect (Object)
Effect (Standard)

**Fig. 3.17** Systemic model of the self-optimizing development process

process. It is checked, if the project plan and the internal objectives of the development process are underachieved, achieved, or overachieved. On the basis of this information alternative process plans can be generated, if necessary.

2. **Determining the system of objectives:** The objectives and their weighting can be changed on basis of the information collected in the situation analysis. Therefore alternative process plans are evaluated by references to the internal system of objectives. If there is a better process plan than the actual implemented plan, according to the internal system of objectives, an adaptation of the system behavior is initiated.
3. **Adapting the systems behavior:** The adaptation of the behavior of the development process is effected by a new process plan. This plan can have the same structure, but different parameters or both.

For the realization of the self-optimization process a **control system** is necessary. The structure of this system is illustrated in Fig 3.18. It is a network of interacting control loops, consisting of the following components: controller, monitor, planner, objective and implementing determination.

These four components can influence the execution of the development process directly or implicitly through the modification of its system of objectives. In general the four components of the control system can be allocated to the three levels of the Operator-Controller-Module (cf. Sect. 1.3.2).

#### 3.4.1.3 Procedure for the Realization of the Self-optimizing Development Process Planning

For the self-optimizing adaptation of system development processes it is necessary to know possible process sequences for the transformation of the object of development into the complete description of the system documentation. Hence, the objective of the procedure is the automatic generation of Pareto optimal process plans, which each contain a process sequence, a plan for time and costs as well as for the necessary auxiliaries and resources.

The procedure is knowledge-based. It uses the information modeled in the discipline-spanning **system model** of the object of development and in the process elements (Fig. 3.19). Information specified in the system model, modeled with the specification technique CONSENS (cf. Sect. 4.1), are e.g. attributes of the subsystems, like the responsible discipline or the degree of complexity. For the specification of the process elements the modeling language OMEGA (cf. Sect. 3.1) is used. The process elements are distinguished in basic elements and coordination elements. The basic elements depict the analysis and synthesis of development activities; the coordination elements describe the synchronization and integration activities for the coordination of the discipline- or subsystem-spanning development activities. Information specified within the process elements, are e.g. input- and output-objects, used resources, auxiliaries, or costs.

The procedure is divided into three phases (Fig. 3.19): 1) analysis of the object of development, 2) process synthesis, and 3) scheduling. They result in alternative and Pareto optimal process plans, with which the object of development can be

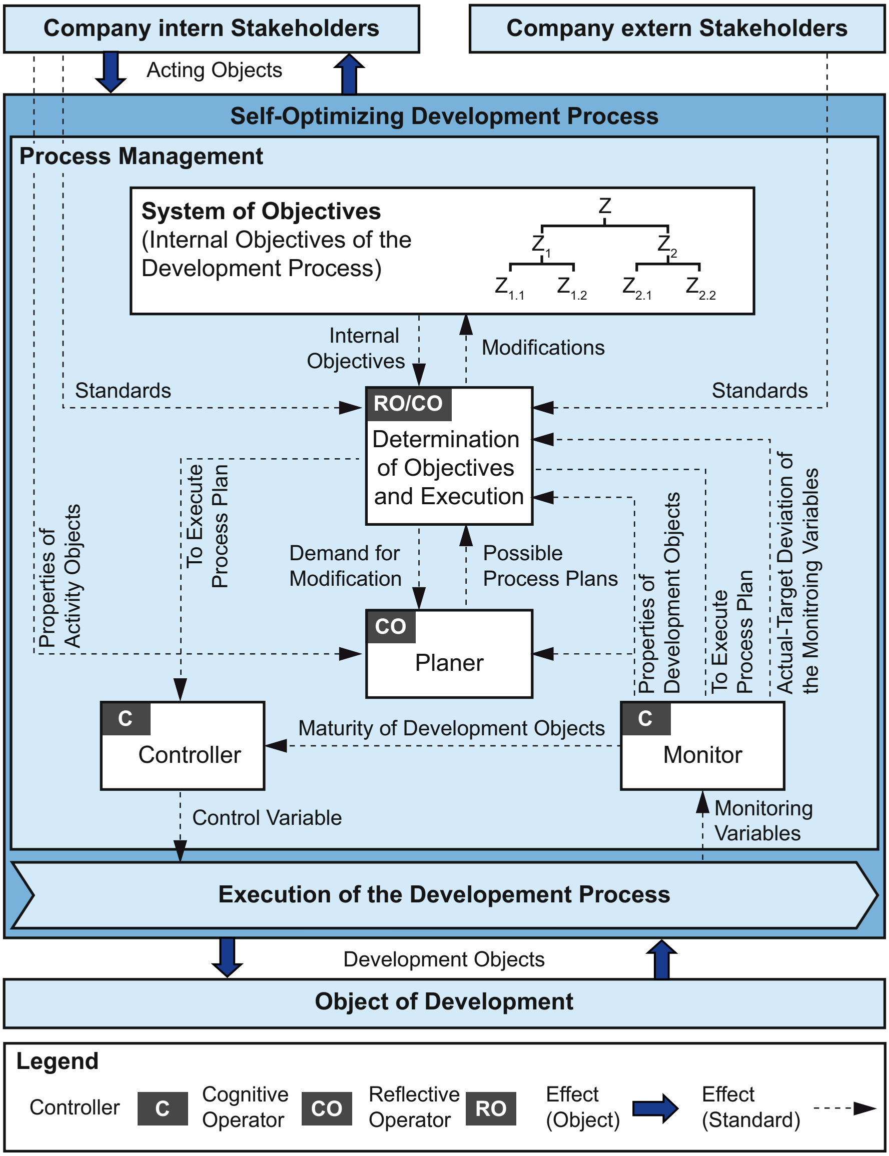


**Fig. 3.18** Control System of the self-optimizing development process

transferred into the complete description of a technical system, fulfilling all objectives. Below, we describe the three phases in more detail.

**Phase 1 – Analysis of the development object:** The objective is a first rough development object oriented structure of the development process. Therefore the subsystems, which have to be developed in parallel, are identified within the system model of the object of development. For each subsystem a swim lane within the

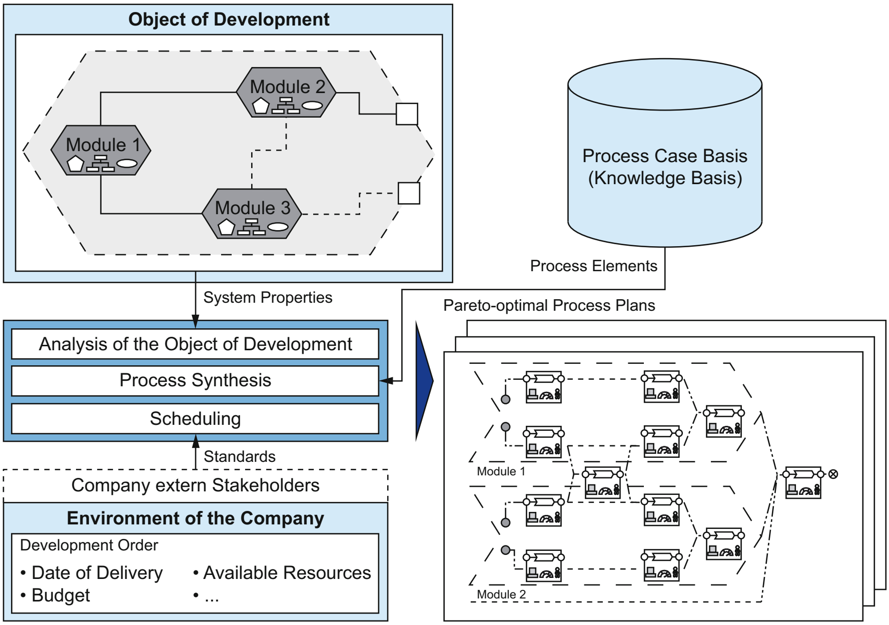
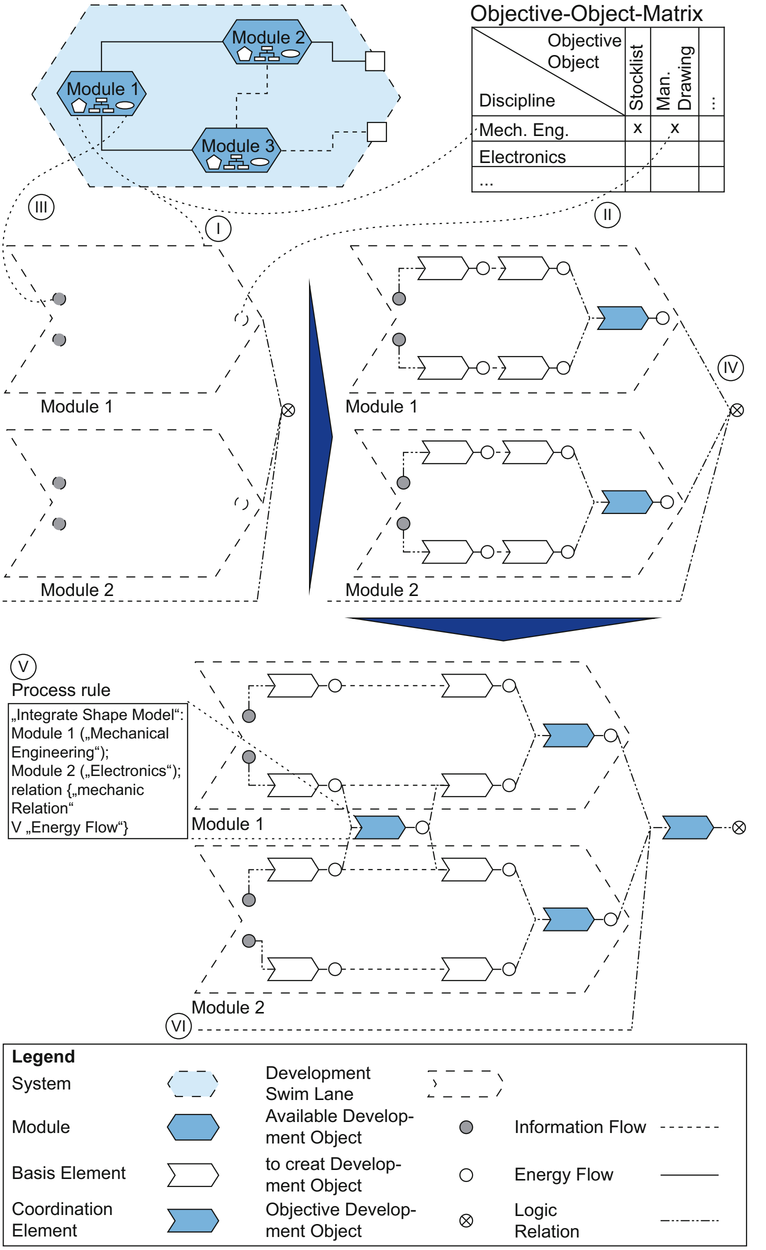


**Fig. 3.19** Components of the procedure for the system model based process planning

development process is generated (*I* in Fig. 3.20). At the end of each swim lane subsystem-specific development objects have to be available. To identify which class of mechatronic systems (e.g. dynamic multibody system, integrated mechatronic system) and to which discipline (e.g. mechanical engineering, software engineering) the system elements of the subsystems belong to, classification trees are used. The required development objects are selected from the so called objective-object matrix (*II* in Fig. 3.20), which allocates disciplines to development objects. For mechanical engineering e.g. production drawings or the subsystem part list have to be available. At the end of phase 1 already available development objects are captured and allocated to the swim lanes (*III* in Fig. 3.20). As a result of the first phase we get subsystem-specific development swim lanes and their input- and output-objects.

**Phase 2 – Process synthesis:** During the process synthesis process elements are combined to possible process sequences, with which the development objects can be transferred into the objective development objects. For the identification of necessary process elements the properties of system elements from the system model and the process elements are compared by classification trees. The suitable process elements are allocated to the subsystem specific swim lanes. Subsequently the process elements are combined to process sequences for each subsystem-specific swim lane, according to their possible input-output-relations (*IV* in Fig. 3.20). This is carried out to a backward breadth-first search. To link these subsystem-specific process sequences we use subsystem-spanning coordination elements, which are



Objective-Object-Matrix

| Objective<br>Object<br>Discipline | Stocklist | Man. Drawing | ⋮ |
|---|---|---|---|
| Mech. Eng. | x | x | |
| Electronics | | | |
| ... | | | |

„Integrate Shape Model“: Module 1 („Mechanical Engineering“); Module 2 („Electronics“); relation {„mechanic Relation“ V „Energy Flow“}

**Fig. 3.20** Phases of the procedure for the system model based process planning

selected according to the modeled relations among the system elements of the system model. The necessary coordination elements are determined by process rules (*V* in Fig. 3.20). Therefore, the following information has to be analyzed: classification of the subsystems, relations among the system elements in the active structure of the system model, and the hierarchical relations of the corresponding functions within the function hierarchy. Next, possible auxiliaries, which are necessary to conduct the process, are identified and allocated to them. For this execution rules (relation between input-object, auxiliary, and output-object) of the process elements are used. If it is possible to realize an input-output-relation through several auxiliaries, alternative process sequences are built. The result is a set of possible process sequences, which depict the logic sequence of the necessary development activities.

**Phase 3 – Scheduling:** In the next step time and cost plans are calculated for the process sequences, considering the applied auxiliaries and resources. Therefore the process sequences are transferred to Hierarchical Precedence Graphs (HPG) according to Klöpper (2010) [36]. Based on the execution rules of the process the abilities of the applied resources are identified. The appropriate resources are selected and allocated to the processes by using a heuristic multiobjective search (cf. [13]), to generate Pareto optimal process sequences [36], [37]. Resources are allocated to the process elements according to key figures (e.g. duration, costs), whereby resources with the best key figures are chosen. At this point, there is rarely a resource, whose performance is better in all key figures. For example: An experienced engineer delivers a higher process quality, but is more expensive then an inexperienced engineer of the same duration. In such conflicts a separate plan is generated for each option. As a consequence one plan e.g. is optimal according to process quality and another according to costs. In this manner all processes sequences are planned, whereby each new generated plan is compared to the already available plans; inferior plans are scrapped. A plan is better than all other plans, if one or more key figures are better than the other and all the rest of the key figures are equal.

This procedure is deployed on all generated possible process sequences in phase 2. The outcomes are **Pareto optimal process plans** of all possible process sequences for the development of the complete system documentation. These are delivered to the objective and implementing determination within the control device of the self-optimizing management of the development project (Fig 3.18), which selects a particular plan according the actual development situation.

#### 3.4.1.4 Procedure for the Situation Specific Determination and Weighting of Development Process Objectives

The procedure for the system model based process planning delivers a set of Pareto optimal process plans. From this set an appropriate plan has to be chosen according to the actual development situation. This requires a basis of decision making, which is the system of objectives for the development process. The following procedure is used to determine and weight the objectives of the development process in a specific development situation. From this the following aspects are considered: development situation, external stakeholders, environment of the company, development object, previous sequence of the development process. The sequence of the procedure consists of five phases. Its objective results in the situation specific internal system of objectives of the development process. According to the influence, changing the development situation, passed each or only single phases of the procedure.

**Phase 1 – Determination of internal system of objectives:** The target is the determination of the basic quantity of external and inherent objectives, which forms the potential internal objectives of the development process. The external objectives result from the available development order, the chosen strategy options of the company, and the actual properties of the development object. Exemplary external objectives are: "minimize development time", "minimize development costs", or "minimize technical risk". The inherent objectives are experiences of good engineer-related practice. Examples are: "minimize personnel turnover", or "minimize media disruption". All external and inherent objectives are taken together and broken down to the swim lanes of the subsystems appropriate to the properties of the actual available development objective.

**Phase 2 – Determination of strategic priority of objectives:** In this phase the priority of the objectives is determined forming a strategic view point. The identified external und inherent objectives (phase 1) are compared by pairs considering the development order and the strategy option. It is stated by pairs, if an objective has the same, a lower or greater relevance than the other concerning the development project. The calculated strategic priority of all objectives forms the fundamental weighting of the external and the inherent objectives form the view points of the environment of the company and the external stakeholders.

**Phase 3 – Determination of technological priority of objectives:** The technological priority of the objectives is calculated, to consider the fact that the attributes of the subsystems influence the objectives of the development process. First the technological relevance of each subsystem is calculated, which can be deduced from the complexity and the dynamic index of a subsystem. The dynamic index describes, how strong a subsystem is influenced from other subsystems or if it even influences them. Second it is calculated, how the technological relevance of a subsystem affects the objectives of the swim lane. This is set depending on whether the technological relevance correlates negatively or positively with the identified external and inherent objectives, identified in phase 1. A combination of both, the technological relevance and their correlation with the objectives, form the technological priority for each objective per subsystem-specific swim lane.

**Phase 4 – Determination of demand for modification of objectives:** For the situation specific weighting of the objectives, it is necessary to identify the deviation from the project plan and the resulting demand to modify the objectives. For this purpose monitoring variables are stated (e.g. depleted budget, progress of the development object, elapsed time). The next step is the definition of the impact (positive, negative, and neutral) of the actual-target-deviation of the monitoring variables on the weighting of one or more objectives. Out of this the demand for modification is determined for each objective and each subsystem.

**Phase 5 – Weighting of internal objectives:** The weighting of the internal objectives for each subsystem specific swim lane are calculated on the basis of the strategic (phase 2) and the technological priority (phase 3) of the objectives as well as the actual demand for modification (phase 4). Therefore a weighting function is used, which considers the risk of the development project as the slope. From the weighted function and the demand for modification of the objectives the relevance per objective and subsystem is calculated. Through normalization of the relevance's per subsystem we get the weighting of the internal objectives per subsystem. The described procedures deliver 1) Pareto optimal development processes for the actual development situation and 2) the situation specific internal system of objectives of the development process. On this basis the optimal plan has to be selected from the set of all possible plans according to the development situation. On this, two steps are necessary: 1) Determination of plan specific objective fulfillment and 2) plan selection.

### *3.4.2 Systematic Planning of Synchronizations*

One particular challenge for the planning of an appropriate development process is the interleaving of the domains involved. Models of the domains overlap or interfaces between domain models exist, which lead to dependencies of the domains. If the process does not consider these dependencies, inconsistencies and costly iterations are likely to occur. Appropriate planned synchronizations of the domain-specific processes can prevent these problems.

Most existing approaches for process planning and modeling require a prior knowledge of all synchronizations. Examples are imperative process modeling approaches such as petri nets [4], or rule-based approaches. Usually, organizations are aware of recurring synchronizations, but lack knowledge on synchronizations that are highly dependent on the dependencies of the domains' models, i.e. synchronizations depending on the structure of the system.

Data-centric approaches address the interplay of the process and the domains' models and provide better support to consider dependencies of the domains' models. Müller et al. (2007) specify subprocesses for each system element in a state-based manner [40]. Transitions between the states represent the process steps. Similarly, process-centered environments use state-based languages to specify executable process step for each domain model [20, 33, 39, 41]. Still, process engineers must know the states and interactions of the different domains' models.

To overcome these problems, we will introduce a systematic approach that enables project managers to derive synchronizations based on known consistency relations of the domains' models. Although engineers are usually not aware of the detailed interactions between the domains, most engineers have local knowledge, i.e. they know to which modeling elements their domain models must be consistent. For example, the durations for the exchange of signals of the controller play an important role for the behavior of the discrete software. A software engineer knows that the time constraints of the discrete software must correspond to these durations of the controller. However, the software engineer is not aware of the process steps that a control engineer must perform to define the durations.

The systematic approach suggests processes based on a set of unrelated subprocesses for the domains and dependencies between the domain models. In order to derive initial subprocesses, project managers can apply the framework for a self-optimizing process (cf. Sect. 3.4.1) and use the frameworks result as an initial process. We, therefore, assume that the subprocesses are known before the planning of synchronizations. The focus of the systematic planning approach lies on the specification of the **consistency relations** and their interplay with the subprocesses to derive processes that consider all necessary synchronizations.

As illustrated in Figure 3.21, the approach consists of five steps: (1) a process manager specifies the interplay of the process and the dependencies of the models, while project managers specify the consistency relations among the models that hold for the project at the planning time, (2) a set of processes is generated automatically based on the consistency relations and the domain-specific processes, and the set is sorted based on evaluation heuristics that are applied to estimate the process quality, (3) a project manager selects one process that fits best for the project, (4) engineers perform the process to develop the self-optimizing system, and (5) process engineers evaluate the executed process and adjust the evaluation heuristics. A project manager typically applies these steps after the conceptual design and repeats the steps, each time the dependencies of the models or the domain-specific processes change.

Vital for the approach is a process modeling language that considers the specification of the models' dependencies (step 1) and the synthesis of the synchronization (step 2). We illustrate both with the help of an scenario that is based on our experience on the development of the RailCab. We will introduce the example in the next section. Afterwards, we will explain the two steps in separated sections.

### 3.4.2.1 Example Scenario

The interaction of the domain control engineering and software engineering during the design of the convoy maneuver of the RailCab (cf. Sect. 2.1) serves as a running example in the following. In order to keep a constant distance while participating in a convoy, RailCabs use a distance controller that controls the speed of the RailCab. For safety purposes, RailCabs exchange their position to calculate the distance to the next RailCab in addition to distance sensors. Control engineers model the control loop of the distance controller. In parallel, software engineers derive a component model and design the message-based communication behavior for all components. In particular, software engineers specify the exchange of the position values between RailCabs. The domain-specific processes must be integrated by defining synchronizations [31]. As an example of synchronizing two domains, we focus on the exchange of message time intervals, which are necessary for the specification of the communication behavior.

Controllers are usually specified in terms of differential equations (step ”Finalize Controller”). The differential equations are evaluated with a fixed sample rate that is defined by control engineers. This sample rate influences the stability of the

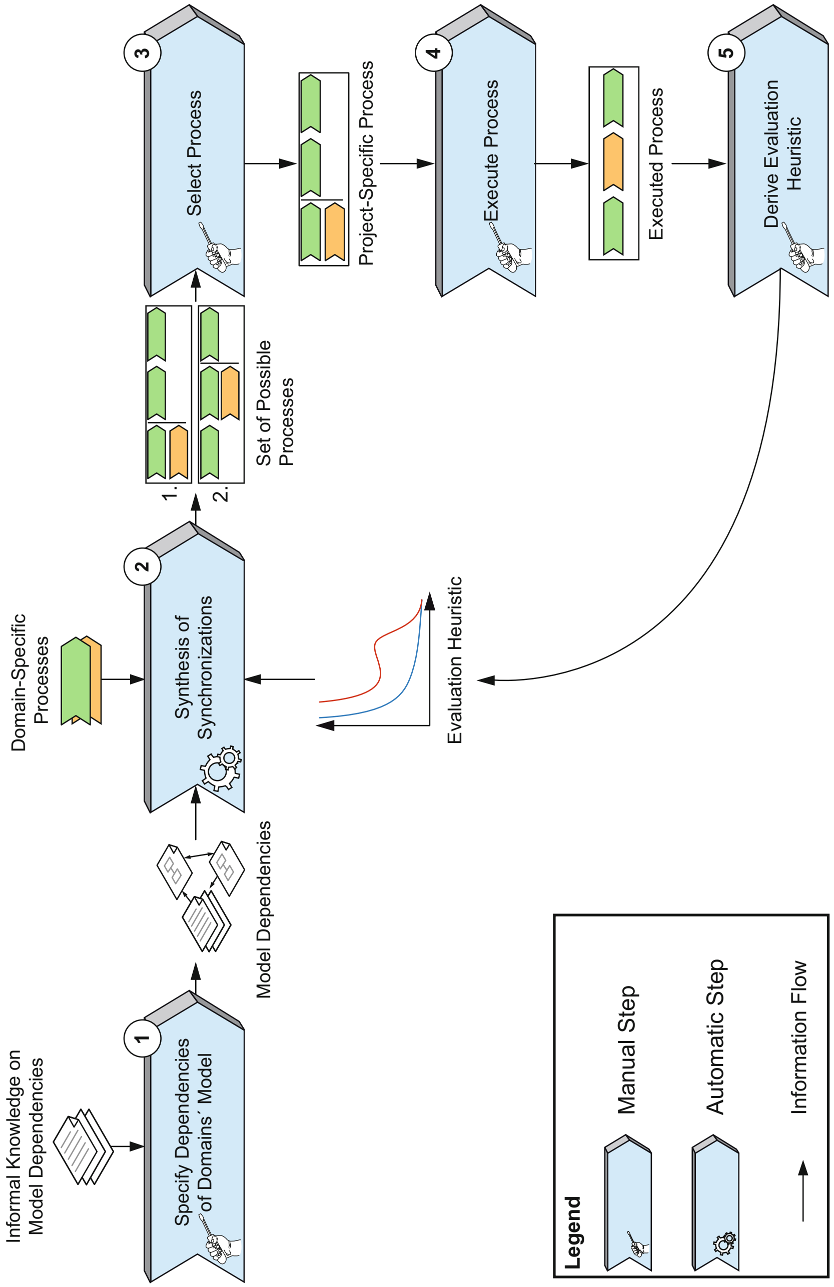

Informal Knowledge on Model Dependencies
1
Specify Dependencies of Domains' Model
Model Dependencies
Domain-Specific Processes
2
Synthesis of Synchronizations
Evaluation Heuristic
1.
2.
Set of Possible Processes
3
Select Process
Project-Specific Process
4
Execute Process
Executed Process
5
Derive Evaluation Heuristic
Legend
Manual Step
Automatic Step
Information Flow

**Fig. 3.21** Steps to plan synchronizations systematically

control strategy. Stability means that a controller can compensate disturbances of the environment without oscillating.

The sample rates are also important for the specification of the communication behavior. If controller signals, e.g. the position values, are transmitted, a current value of a signal must always be available according to a given sample rate [42]. Messages which transmit these signals must be executed within the so-called message time intervals according to the given sample rates. Hence, software engineers must wait to specify the coordination patterns in the step ”Determine Coordination Patterns” (cf. Sect. 3.3.3) until control engineers have defined the sample rates. We identified two cases where control engineers hand-over the message time intervals.

First, the message time intervals may already be known at the time when the principle solution is developed. This is the case, if (partial) models or parts thereof are being reused in new projects. Especially, controllers can be reused so that the required information for software engineering can be added to the principle solution. This removes the dependencies of the process steps and enables a parallel development of both disciplines. Note, that we assumed this case for the reference process (cf. Sect. 3.3.3).

Second, software engineers may have to wait until the control engineers designed the controller and ensured its stability in step ”Analyze Closed-Loop System” (cf. Sect. 3.3.2). This is the case, if control engineers model new sophisticated controllers, in order to implement innovative functionality. During the interdisciplinary conceptual design, control engineers can only estimate the message time intervals. It is, therefore, likely that the message time intervals change during the controller design.

The goal of our approach is to identify these cases automatically and add the synchronization that fits best for the current project. We will demonstrate in the next two sections how our approach can be applied to the RailCab example.

### 3.4.2.2 Combined Modeling of Process and Domain Model Dependencies

Process engineers and project managers have to capture the domain-specific processes, the domain models, and their consistency relations in a computer interpretable representation to derive the synchronizations. We extend Yet Another Workflow Language (YAWL) [3], a business process language which is inspired by petri nets, to consider consistency relations of the domain models.

In contrast to data-centric processes, we do not specify the dependencies of the process steps or the domain models explicitly. Instead, we introduce two new modeling languages: (1) **artifact consistency graphs** (ACG) to specify the consistency relations of the domain models that hold currently, and (2) **artifact story diagrams** (ASD), which describe changes to the consistency relations during process execution. This has the advantage, that information on consistency relations and the changes of these relations is usually available.

An **ACG** abstracts the domain models and their relations to capture only the information that is relevant to derive synchronizations. Concerning the relations, we identified three common relations: modeling elements can (1) be consistent, (2)

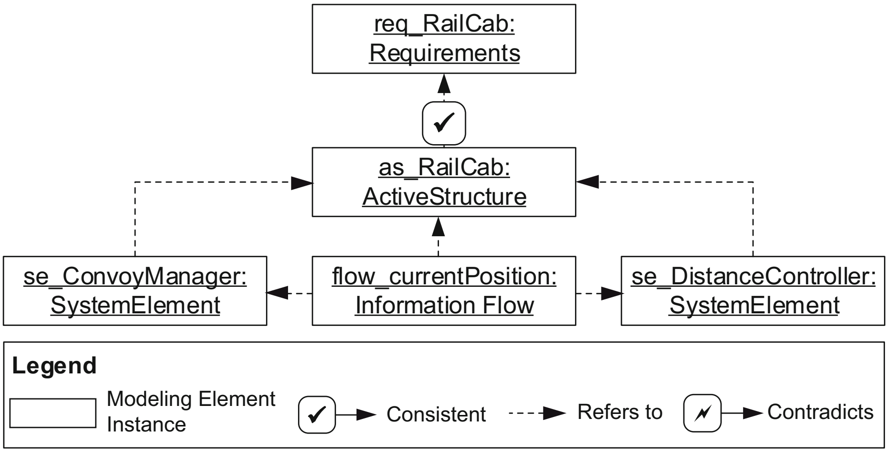


**Fig. 3.22** Artifact Consistency Graph (ACG) of the development state after the domain-spanning conceptual design

contradict, or (3) refer to each other. Initially, a process manager must collect the relevant modeling elements. For instance, message time intervals play an important role for the process planning, but the differential equations of the controllers are not important. As a rule of thumb, modeling elements that overlap with modeling elements of other domains are usually relevant. Thereafter, project managers can derive an ACG of the current development situation from the domain models and based on interviews with the engineers.

Figure 3.22 shows an example of an ACG that represents the state of the consistency relations after the conceptual design for the RailCab convoy. The active structure of the RailCab is consistent with the requirements of the RailCab. The relation *refers to* describes relevant information on the structure of the models. In our example, the active structure consists of two important system elements: a system element *se_ConvoyManager* that encapsulates the communication behavior of the convoy, and a system element *se_DistanceController* that represents the distance controller. These system elements are connected by an information flow.

If an engineer performs a process step, the existing modeling elements and their consistency relation may change. For instance, the step ”Finalize Control Strategy” adds a controller and its message time intervals. Artifact story diagrams (ASD) are a compact modeling formalism to capture such changes. A process engineer specifies a set of ASD for each process step after the collection of relevant modeling elements. Process engineers only have to do this once, in contrast to the specification of the ACG that project managers must specify each time synchronizations are planned.

**ASDs** base on Story Diagrams (SD) [19], which were developed to describe software behavior in a graphical manner. An ASD describes which modeling elements and relations must exist in an ACG before a process step can be performed and describes which modeling elements and relations are created or removed during the

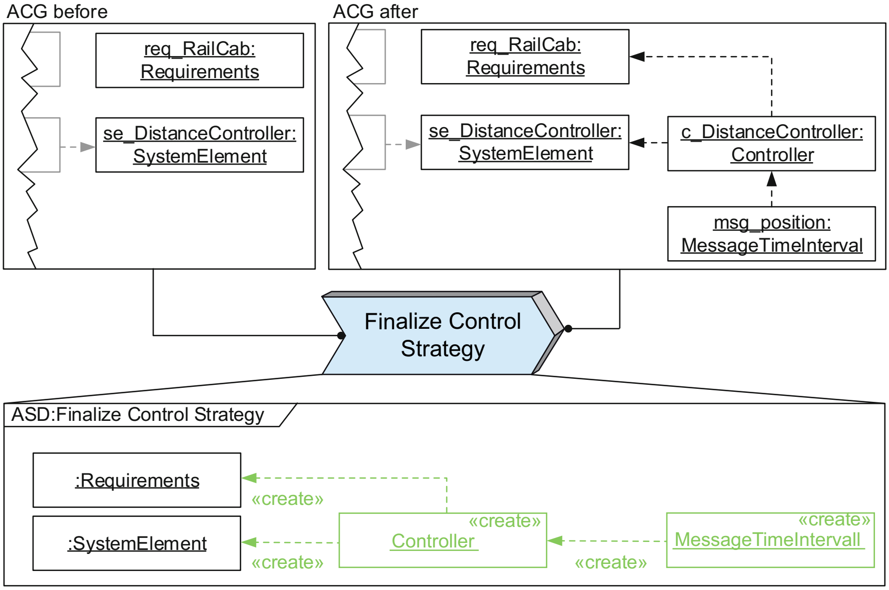

ACG before
req_RailCab: Requirements
se_DistanceController: SystemElement
ACG after
req_RailCab: Requirements
se_DistanceController: SystemElement
c_DistanceController: Controller
msg_position: MessageTimeInterval
Finalize Control Strategy
ASD:Finalize Control Strategy
:Requirements
:SystemElement
«create»
«create»
«create»
Controller
«create»
MessageTimeIntervall
«create»

**Fig. 3.23** Example of an Artifact Story Diagram (ASD) to specify the interaction of process steps and Artifact Consistency Graphs (ACG)

process step. If a process step has more than one ASD, the step is executable, if at least one ASD can be applied.

The bottom of Figure 3.23 shows the ASD for the process step ”Finalize Control Strategy”. The annotation «create» means that a new modeling element or a new relation will be created. Similarly, elements or relations are removed, if they have the annotation «delete».

The ASD can be applied if a system element and requirements exist in the ACG. Then, a controller (*c_DistanceController*), its message time intervals, and two *refers to* relations are added. Note that the *c_DistanceController* does not yet have a *consistent* relation to the system element and the requirements, because the control engineers have not checked if the controller fulfills the requirements.

After this specification step, we can analyze the dependencies between process steps of different domains in the current process and add synchronizations accordingly.

### 3.4.2.3 Synthesis of Synchronizations

The aim of the synthesis of synchronization is an overall development process that contains all necessary synchronizations between the domains. Inputs for the approach are the set of ASD, the inital ACG, and a set of domain-specific processes that are tailored for the current project. We assume that the domain-specific processes are separated by process specifications, i.e. an overall process does not exist.

Initially, we create an overall process that consists of a parallel composition of the domain-specific processes.

In order to find out where synchronizations are necessary, we simulate the overall process. If the process lacks synchronizations, one of the following two situations occur during the simulation: (1) parallel processes of the domains produce conflicting modeling elements or consistency relations, or (2) a process step requires modeling elements with consistency relations that are produced by another domain. We add synchronizations between the process steps that depend on each other.

Figure 3.24 shows an example of a simulation run. The top of Figure 3.24 illustrates two process steps of the domain control engineering and corresponding set of ACG. The bottom of Figure 3.24 illustrates two steps of the domain software engineering. The analysis simulated the steps of the domain control engineering and the step ”Derive Component Model” of the domain software engineering. We will see later, that the simulation can not execute the step ”Determine Coordination Pattern”, because a necessary synchronization between the domains is missing.

The $ACG_{ConceptualDesign}$ represents the consistency relations after the ”Domain-spanning Conceptual Design”. This ACG forms the basis for the subsequent simulation of the subprocesses of the domains software and control engineering. First, control engineers add a new controller and define its message time intervals ($ACG_{FinalizeControl}$). Second, they ensure that the controller meets the requirements, i.e. the controller is stable. As a consequence, consistency relations are added to the ACG. The result is the $ACG_{AnalyzeSystem}$.

Independently, we simulate the process of the domain software engineering. During the step ”Derive Component Model”, software engineers add components for each system element and connectors for each information flow of the active structure. The result is the $ACG_{ComponentModel}$. The step ”Determine Coordination Pattern” requires message time intervals of a stable controller, i.e. the controller has to be consistent with the requirements. The message time intervals are, however, not defined during the software design. Therefore, the simulation can never execute this step.

An analysis searches the results of all steps that are executed in parallel to ”Determine Coordination Pattern” for the missing controller with a consistency relation to the requirements and the message time intervals. As the step ”Analysis Closed-Loop System” provides these, a synchronization after the step ”Analysis Closed-Loop System” and before the step ”Determine Coordination Pattern” is added. This is, however, not the only solution. Each process step after ”Analyze Closed-Loop System” also provides the required message time intervals.

Generally, the simulation starts with the initial ACG, as specified by the project manager. A process step can be executed, if it is executable according to the domain-specific processes, and if it has at least one ASD that can be applied to the current ACG. A process step execution includes the execution of the ASD. As a consequence, the ACG is modified. Hence, the ACG evolves during the simulation of the process.

If process steps produce conflicting modeling elements or relations, we add a new synchronization between these steps. If a process step waits for a combination

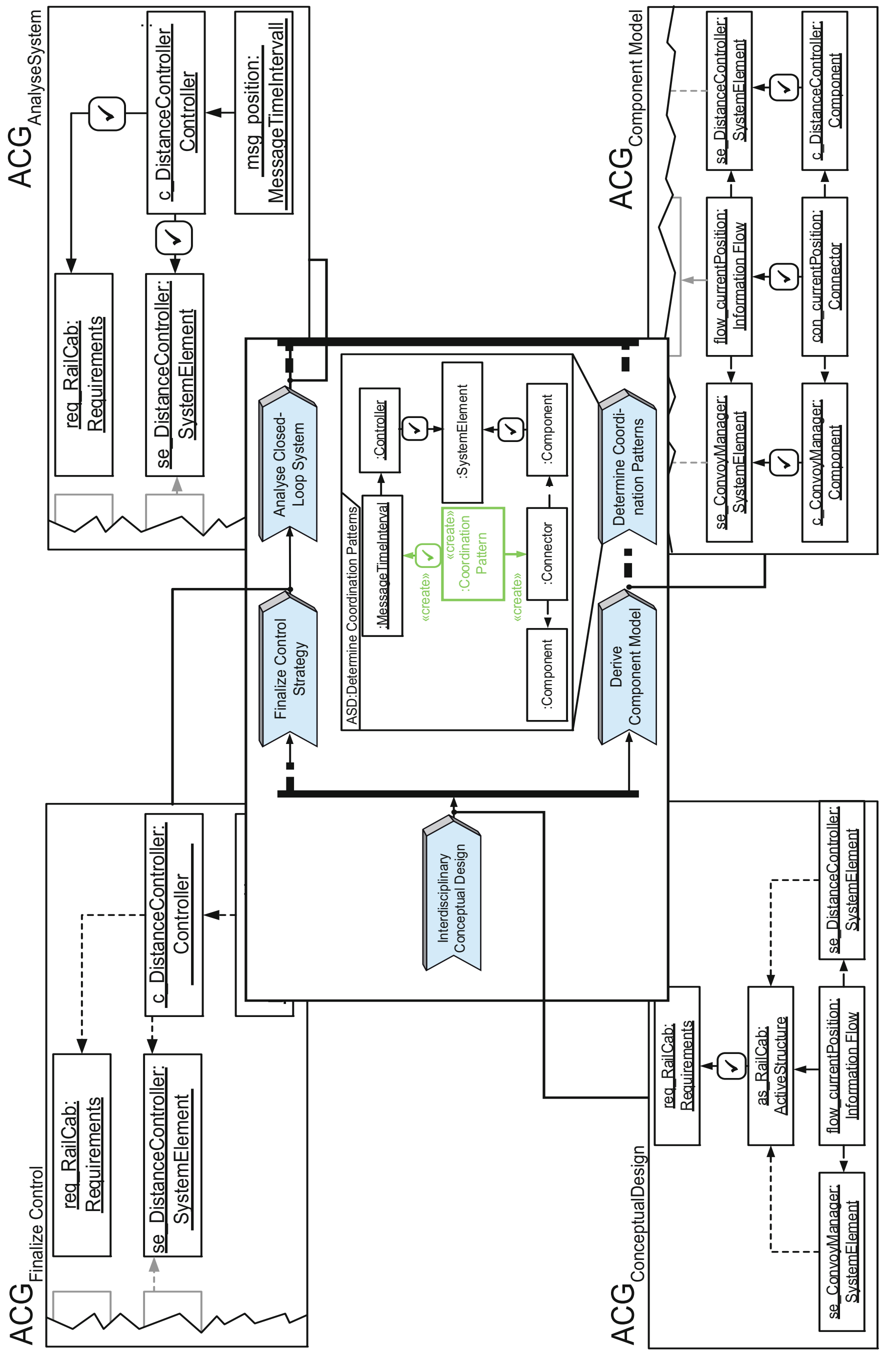


**Fig. 3.24** Example of a synchronization identified by a deadlock

of modeling elements and consistency relations from another domain, we add a new synchronization between the step that waits and the process steps that provide the necessary modeling elements and consistency relations. Usually, there are more than one combination of steps that provide the required modeling elements and relations. As described above, the message time intervals are available after the step ”Finalize Control Strategy”, but they are still available after the following step, e.g. ”Analyze Closed-Loop System”. Because the step ”Analyze Closed-Loop System” ensures the stability of the controller, a synchronization after this step is the better choice. We, therefore, create different variants of the process for each possible combination.

We apply evaluation heuristics to sort the process variants. We have identified three common heuristics: (1) before modeling elements are handed-over, they should be analyzed, (2) synchronizations due to missing modeling elements or consistency relations should be performed as early as possible in the process, (3) a synchronization should occur with the least possible process steps.

## References

1. VDI 2206 - Entwicklungsmethodik für mechatronische Systeme. Beuth Verlag, Berlin (2004)
2. Uml 2.2: Superstructure specification (2009)
3. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. Tech. Rep. FIT-TR-2003-04, Queensland University of Technology, Brisbane (2003)
4. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
5. Albers, A.: Five Hypotheses About Engineering Processes and Their Consequences. In: Proceedings of the 8th International Symposium on Tools and Methods of Competitive, Ancona (2010)
6. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdhlking, I., Angel, S.: A Pattern Language. Oxford University Press, Oxford (1977)
7. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
8. Bauer, F., Anacker, H., Gaukstern, T., Gausemeier, J., Just, V.: Analyzing the Dynamical Behavior of Mechatronic Systems Within the Conceptual Design. In: Proceedings of the 18th International Conference on Engineering Design, Copenhagen, pp. 329–336 (2011)
9. Becker, S., Brenner, C., Brink, C., Dziwok, S., Heinzemann, C., Löffler, R., Pohlmann, U., Schäfer, W., Suck, J., Sudmann, O.: The MechatronicUML Design Method - Process, Syntax, and Semantics. Tech. Rep. tr-ri-12-326, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn (2012)
10. Bender, K.: Embedded Systems - Qualitätsorientierte Entwicklung. Springer, Heidelberg (2005)
11. Bengtsson, J.E., Yi, W.: Timed Automata - Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)

12. Burmester, S., Giese, H., Oberschelp, O.: Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In: Braz, J., Araújo, H., Vieira, A., Encarnacao, B. (eds.) Informatics in Control, Automation and Robotics I. Springer, Heidelberg (2006)
13. Dasgupta, P., Chakrabarti, P., Desarkar, S.C.: Multiobjective Heuristic Search - An Introduction to Intelligent Search Methods for Multicriteria Optimization. Vieweg & Sohn Verlagsgeschellschaft, Braunschweig (1999)
14. Dorociak, R.: Early Probabilistic Reliability Analysis of Mechatronic Systems. In: Proceedings of the Reliability and Maintainability Symposium (2012)
15. Dorociak, R., Gaukstern, T., Gausemeier, J., Iwanek, P., Vaßholz, M.: A Methodology for the Improvement of Dependability of Self-optimizing Systems. Production Engineering - Research and Developement 7(1), 53–67 (2013)
16. Dziwok, S., Just, V., Schierbaum, T., Schäfer, W., Trächtler, A., Gausemeier, J.: Integrierter Regelungs- und Softwareentwurf für komplexe mechatronische Systeme. In: Tagungsband vom Wissenschaftsforum 2013 Intelligente Technische Systeme - 9, Paderborn. Paderborner Workshop Entwurf mechatronischr Systeme (2013)
17. Eckardt, T., Heinzemann, C., Henkler, S., Hirsch, M., Priesterjahn, C., Schäfer, W.: Modeling and Verifying Dynamic Communication Structures Based on Graph Transformations. Computer Science - Research and Development 28, 3–22 (2013)
18. Ehrlenspiel, K.: Integrierte Produktentwicklung, 2nd edn. Carl Hanser Verlag, München (2007)
19. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams - A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) Graph Transformation. LNCS, vol. 1764, pp. 296–309. Springer, Heidelberg (2000)
20. Fuggetta, A.: Software Process: A Roadmap. In: Proceedings of the Conference on The Future of Software Engineering, pp. 25–34. ACM Press, New York (2000)
21. Gajski, D., Kuhn, R.: Guest Editors Introduction: New (VLSI) Tools. Computer 16(12), 11–14 (1983)
22. Gausemeier, J., Frank, U., Donoth, J., Kahl, S.: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme des Maschinenbaus - Teil 2. Konstruktion 9 (2008)
23. Gausemeier, J., Frank, U., Donoth, J., Kahl, S.: Specification Technique for the Description of Self-optimizing Mechatronic Systems. Research in Engineering Design 20(4), 201–223 (2009)
24. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
25. Gausemeier, J., Schäfer, W., Greenyer, J., Kahl, S., Pook, S., Rieke, J.: Management of Cross-Domain Model Consistency During the Development of Advanced Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
26. Giese, H., Tichy, M.: Component-based Hazard Analysis: Optimal Designs, Product Lines, and Online-reconfiguration. In: Proceedings of the 25th International Conference on Computer Safety, Security and Reliability, Gdansk (2006)
27. Giese, H., Tichy, M., Burmester, S., Schäfer, W., Flake, S.: Towards the Compositional Verification of Real-time UML Designs. In: Proceedings of the 9th European Software Engineering Conference Held Jointly with the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Helsinki, pp. 38–47. ACM Press, New York (2003)

28. Greenyer, J., Rieke, J., Schäfer, W., Sudmann, O.: The Mechatronic UML Development Process. In: Tarr, P.L., Wolf, A.L. (eds.) Engineering of Software, pp. 311–322. Springer, Heidelberg (2011)
29. Heinzemann, C., Pohlmann, U., Rieke, J., Schäfer, W., Sudmann, O., Tichy, M.: Generating Simulink and Stateflow Models From Software Specifications. In: Proceedings of the 12h International Design Conference DESIGN, Dubrovnik (2012)
30. Heinzemann, C., Priesterjahn, C., Becker, S.: Towards Modeling Reconfiguration in Hierarchical Component Architectures. In: Proceedings of the 15th ACM SigSoft International Symposium on Component-Based Software Engineering, Bertinoro, pp. 23–28 (2012)
31. Heinzemann, C., Sudmann, O., Schäfer, W., Tichy, M.: A Discipline-spanning Development Process for Self-adaptive Mechatronic Systems. In: Proceedings of the International Conference on Software and System Process, San Francisco (2013)
32. Isermann, R.: Mechatronische Systeme - Grundlagen. Springer, Heidelberg (2008)
33. Junkermann, G., Peuschel, B., Schäfer, W., Wolf, S.: MERLIN - Supporting Cooperation in Software Development Through a Knowledge-based Environment, pp. 103–129. Research Studies Press Ltd., Taunton (1994)
34. Kahl, S., Gausemeier, J., Dumitrescu, R.: Interactive Visualization of Development Processes. In: Proceedings of the 1st International Conference on Modelling and Management of Engineering Processes (2010)
35. Kahl, S.M.: Rahmenwerk für einen selbstoptimierenden entwicklungsprozess fortschrittlicher mechatronischer systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagschriftenreihe, Band 308, Paderborn (2013)
36. Klöpper, B., Ishikawa, F., Honiden, S.: Service Composition with Pareto-optimality of Time-dependent QoS Attributes. In: Proceedings of the 8th International Conference on Service-Oriented Computing, Berlin, pp. 635–640 (2010)
37. Klöpper, B., Pater, J.P., Honiden, S., Dangelmaier, W.: A Multi-objective Evolutionary Approach to Scheduling for Evolving Manufacturing Systems. Evolving Systems 3, 31–44 (2012)
38. Köckerling, M.: Methodische Entwicklung und Optimierung der Wirkstruktur mechatronischer Produkte. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagschriftenreihe, Band 143, Paderborn (2004)
39. Künzle, V., Reichert, M.: PHILharmonicFlows - Towards a Framework for Object-aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice 23(4), 205–244 (2011)
40. Müller, D., Reichert, M., Herbst, J.: Data-driven Modeling and Coordination of Large Process Structures. In: Proceedings of the OTM Confederated International Conference on the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS, Vilamoura, pp. 131–149 (2007)
41. Münch, J., Armbrust, O., Kowalczyk, M., Soto, M.: Software Process Definition and Management. Springer, Heidelberg (2012)
42. Osmic, S., Münch, E., Trächtler, A., Henkler, S., Schäfer, W., Giese, H., Hirsch, M.: Safe Online-Reconfiguration of Self-Optimzing Mechatronic Systems. In: Gausemeier, J., Rammig, F.J., Schäfer, W. (eds.) Tagungsband vom 7, Paderborn. Internationalen Heinz Nixdorf Symposium für industrielle Informationstechnik - Selbstoptimierende mechatronische Systeme: Die Zukunft gestalten, pp. 411–426 (2008)

# Chapter 4
# Methods for the Domain-Spanning Conceptual Design

Harald Anacker, Christian Brenner, Rafal Dorociak, Roman Dumitrescu,
Jürgen Gausemeier, Peter Iwanek, Wilhelm Schäfer, and Mareen Vaßholz

**Abstract.** The development of self-optimizing systems is a highly interdisciplinary task, as several domains are involved. Existing design methodologies do not adress this issue, as they focus on the respective domain; a holistic domain-spanning consideration of the system occurs – if at all – only rudimentally. The partial solutions developed by the respective domains may be optimal from the point of view of this domain. However, it does not automatically mean, that the sum of the optimal domain-specific solutions forms the best possible overall solution: "the whole is more than the sum of its parts". This especially holds true for the early design phase, the conceptual design. Its result is the so-called principle solution, which is further refined in the domain-specific design and development. Thus, a great need for methods arises which support the domain-spanning conceptual design for self-optimizing systems in a holistic manner. In this chapter we will introduce such methods. In particular, we will explain the specification technique for the domain-spanning description of the principle solution of a self-optimizing system. Furthermore, methods are explained which support the creation of the principle solution. This includes a method to ensure the consistency of application scenarios, a method for the design of the system of objectives, which is crucial for a self-optimizing system, as well as a method for the re-use of proven solutions for recurring problems (solution patterns). Finally, some analysis methods are explained that are performed on the specification of the principle solution. These are: the early analysis of the reliability and the analysis of the economic efficiency.

The development of self-optimizing systems is structured into the domain-spanning conceptual design and the domain-specific design and development as explained in Chap. 3. During the conceptual design, experts from the domains of mechanical, electrical/electronic, control and software engineering work together and develop the principle solution. The involvement of the different domains in the

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*,
Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_4, © Springer-Verlag Berlin Heidelberg 2014

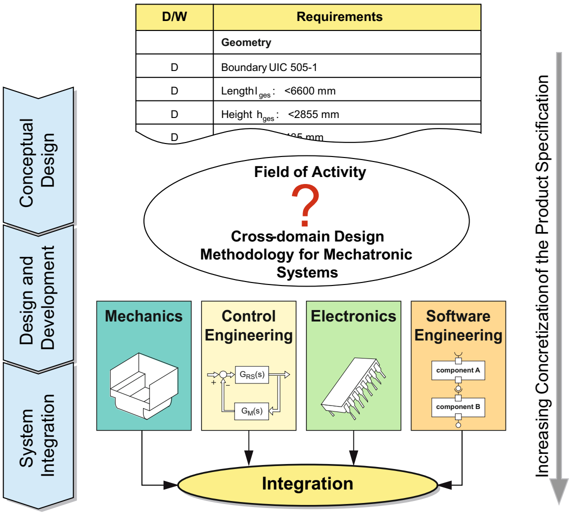


**Fig. 4.1** Central challenge: a specification technique for the description of the principle solution of a self-optimizing mechatronic system [24]

development process for self-optimizing systems as well as the integration of partial intelligence in self-optimizing systems call for new development methods as well as new development tools. Existing design methodologies need to be fundamentally extended. This especially concerns the conceptual design. Certainly, the basic structure of the phases of existing design methodologies (formulation of requirements, definition of functions, etc.) [43] also applies for self-optimizing systems. Nevertheless, such aspects as domain-spanning understanding, modeling of application scenarios, partial intelligence and system behavior have to be considered as well. Due to the involvement of different domains, devices have to be provided which allow fundamental understanding of the whole system by all developers from the very beginning of the development process. The gap between the list of requirements, which is more or less a rough specification of the total system and, hence, leaves much space for interpretation, and well-established specification techniques of the domains involved needs to be closed (Fig. 4.1) [24]. Otherwise, time and cost intensive iterations and failure can emerge when the engineers have to integrate their results in the domain-specific design and development.

To overcome these challenges a holistic description of the principle solution for the whole system is necessary. It describes the basic structure and operational mode of the system, as well as its desired behavior. Moreover, it considers different aspects such as environment, requirements and application scenarios - just to name a few. These different aspects form a coherent system as all aspects correlate with each other. To secure the overall consistency of the principle solution is a challenge, which can only be overcome with an adequate software support. Hence, a software tool which supports the modeling of the principle solution is also a necessary.

A detailed analysis of the state of the art has shown that there are a number of approaches for the specification of mechatronic systems [24]. None of these approaches, however, fulfill the aforementioned requirements to a full extent. In order to address this need for action, we have developed a specification technique CONSENS [8] for the domain-spanning description of the principle solution for self-optimizing systems, which is introduced in the following section [24].

This chapter is structured according to the reference process for the conceptual design (cf. Sect. 3.2). First, the specification technique CONSENS for the domain-spanning description of the principle solution for self-optimizing systems is introduced (Sect. 4.1). Sect. 4.3 - 4.5 explain methods, which support the creation of the principle solution. Sect. 4.3 shows, how to ensure the consistency of application scenarios regarding the discrete behavior specified in them. In Sect. 4.4 it is explained, how the system of objectives, the backbone for self-optimization, is modeled. In Sect. 4.5 we will describe, how solution patterns are used during the conceptual design, i.e. how established solutions for recurring problems can be reused during the specification of the principle solution. The method for the product structuring for self-optimizing systems in the conceptual design is explained in Sect. 4.6. Finally, we will explain how first analyses can be conducted on the description of the principle solution at an early stage. These are the early analysis of the reliability (Sect. 4.7) as well as the analysis of the economic efficiency (Sect. 4.8).

## 4.1 Specification Technique CONSENS for the Description of Self-optimizing Systems

Rafal Dorociak, Roman Dumitrescu, Jürgen Gausemeier, and Peter Iwanek

In accordance to the reference process for the conceptual design (cf. Sect. 3.2) the specification technique CONSENS is used to describe the domain-spanning principle solution for the self-optimizing system [18, 24]. The **principle solution** describes the basic structure (e.g. components of the system and interactions between them), operational mode of the self-optimizing system, and its desired behavior. The principle solution forms the basis for the communication and cooperation of the domains involved (e.g. mechanical and software engineering) in the course of the further domain-specific design and development.

The description of the principle solution of self-optimizing systems consists of eight interrelated aspects. As shown in Fig. 4.2 these aspects are requirements, environment, system of objectives, application scenarios, functions, active structure, shape and behavior. The aspects are computer-internally represented as partial models. The aspects relate to each other and ought to form a coherent system. We will describe each aspect in the following.

**Environment:** There are many interrelations between the system and its environment. Therefore, it is important to analyze the environment of the system to ensure that the final system will work properly in it, without any restrictions caused by not considered interactions. For this purpose the specification technique CONSENS offers the aspect environment. This aspect describes the embedding of the system within its environment; the system itself is treated as a "black-box". In particular,

[8] CONceptual design Specification technique for the ENgineering of complex Systems.

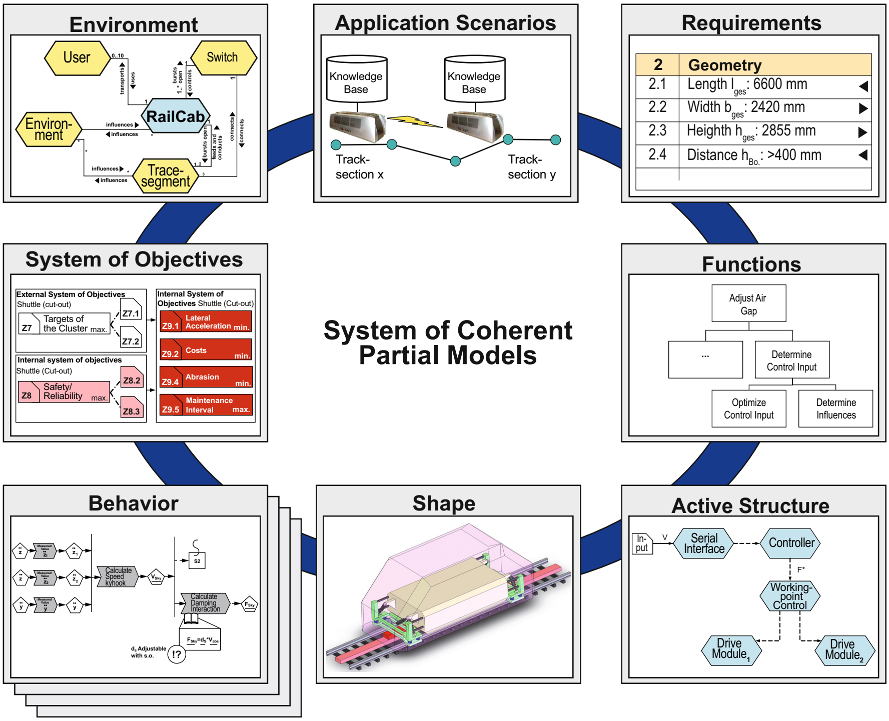


**Fig. 4.2** Partial models for the domain-spanning description of the principle solution for self-optimizing systems [24]

other elements of the environment (e.g. the user, other technical systems or the underground) and their interrelation with the system are described. Relevant influences of the environment on the system such as weather, temperature and humidity are described as well. Influences which have an disturbing impact on the system operation are marked as such. The identification of the relevant influences is supported by respective catalogues and check lists. The interrelations between the system under development and elements of the environment are represented as flows. In principle, three types of flows can be distinguished: information flows, energy flows, and material flows.

Figure 4.3 shows the specification of the environment of the RailCab. In particular, it is shown that the driving behavior of the RailCab is affected by the weight of users and cargo as well as influences from the environment, the state of the track sections as well as the abrasion of the RailCab itself.

**Application Scenarios:** Application scenarios form first concretizations of the system's behavior. They describe the most common operation modes of the system and the corresponding behavior in a rough manner. Every application scenario describes a specific situation (e.g. start-up, failure of the system or an interaction with

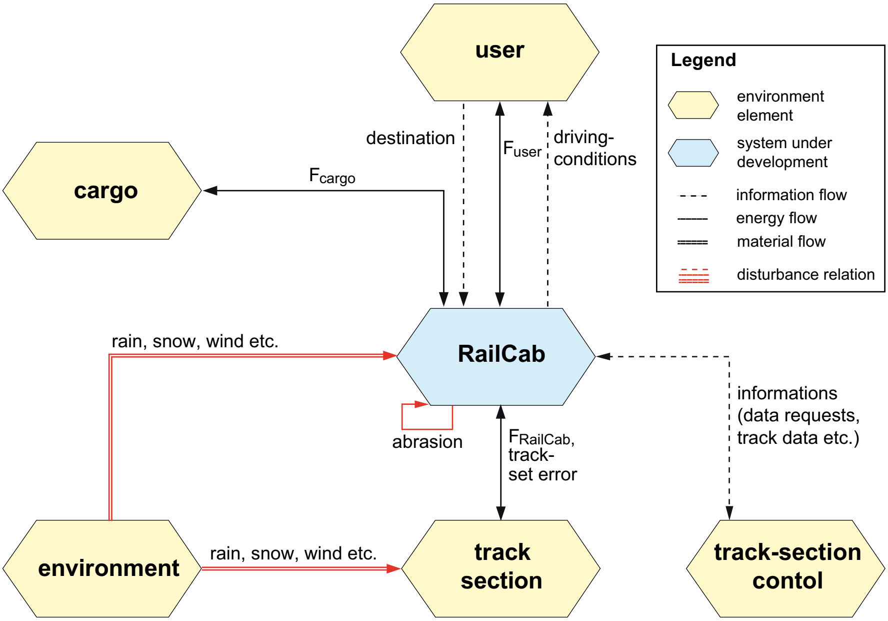
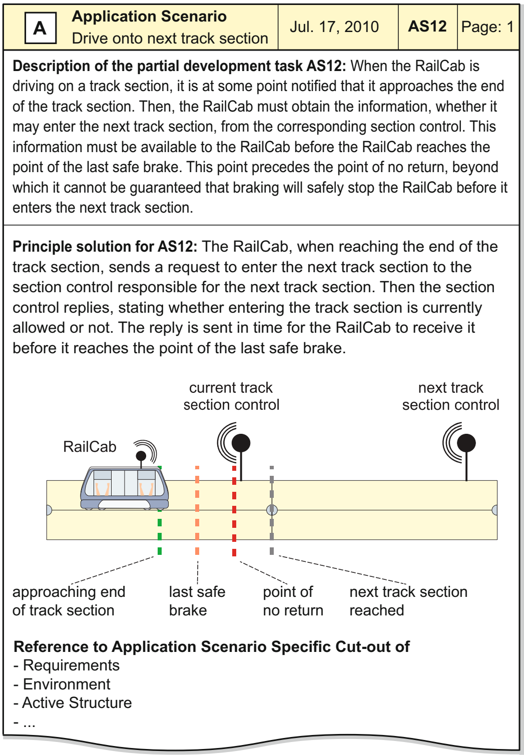


**Fig. 4.3** The partial model environment of the RailCab (excerpt) [24]

the user), and the required behavior of the system for this situation. Thus, application scenarios characterize a problem and the possible solution for it. By modeling the application scenarios requirements and potential operational modes for the system can be identified. Fig. 4.4 shows the application scenario "AS12: Drive onto next track section" for the RailCab as an example. The description of an application scenario includes e.g. general information, like a title, an ID, the date of change, and a textual characterization of the application scenario; to gain greater insight into the application scenario a sketch can be added.

**Requirements:** Based on the general problem definition and the aspects environment and application scenarios, the requirements for the system under development can be defined and modeled. Requirements present an organized collection of requirements that need to be fulfilled by the system under development (e.g. features of the system, overall size, performance, quality). Requirements allow the engineering team to expose what is expected from the future system. They form a corner pillar for the validation and verification in further development phases. Requirements are represented in tabular form; the requirements list. requirements can be decomposed into sub-requirements to structure multiple requirements. For example; "height", "length" and "width" can be sub-requirements of the "size" of an element. Each requirement in the requirements list has an ID, is verbally described, and, if possible, concretized by corresponding parameters (e.g. temperature, length, velocity) and values (e.g. the RailCab should be able to reach a velocity of 100 km/h). Several checklists assist the identification of requirements; see for example

**Fig. 4.4** Description of the application scenario "Drive onto next Track Section" (excerpt) [24]



| | | | | |
|---|---|---|---|---|
| A | **Application Scenario**<br>Drive onto next track section | Jul. 17, 2010 | **AS12** | Page: 1 |

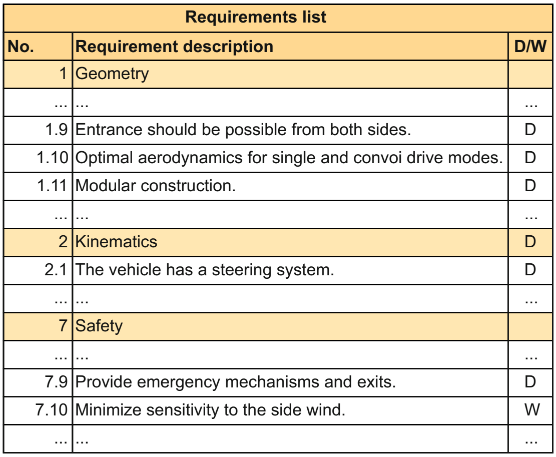
**Description of the partial development task AS12:** When the RailCab is driving on a track section, it is at some point notified that it approaches the end of the track section. Then, the RailCab must obtain the information, whether it may enter the next track section, from the corresponding section control. This information must be available to the RailCab before the RailCab reaches the point of the last safe brake. This point precedes the point of no return, beyond which it cannot be guaranteed that braking will safely stop the RailCab before it enters the next track section.

**Principle solution for AS12:** The RailCab, when reaching the end of the track section, sends a request to enter the next track section to the section control responsible for the next track section. Then the section control replies, stating whether entering the track section is currently allowed or not. The reply is sent in time for the RailCab to receive it before it reaches the point of the last safe brake.

current track section control
next track section control
RailCab
approaching end of track section
last safe brake
point of no return
next track section reached

**Reference to Application Scenario Specific Cut-out of**
- Requirements
- Environment
- Active Structure
- ...

**Fig. 4.5** Requirements on the RailCab (excerpt)



| **Requirements list** | | |
|---|---|---|
| **No.** | **Requirement description** | **D/W** |
| 1 | Geometry | |
| ... | ... | ... |
| 1.9 | Entrance should be possible from both sides. | D |
| 1.10 | Optimal aerodynamics for single and convoi drive modes. | D |
| 1.11 | Modular construction. | D |
| ... | ... | ... |
| 2 | Kinematics | D |
| 2.1 | The vehicle has a steering system. | D |
| ... | ... | ... |
| 7 | Safety | |
| ... | ... | ... |
| 7.9 | Provide emergency mechanisms and exits. | D |
| 7.10 | Minimize sensitivity to the side wind. | W |
| ... | ... | ... |

[11, 43, 47]. Requirements are separated into demands and wishes [43]. If needed, the requirements can also be divided into functional (e.g. the doors of the RailCab should be able to close automatically) and non-functional requirements (e.g. the doors of the RailCab should be red). An excerpt of the requirements list for the RailCab is shown in Fig. 4.5; demands are marked with "D", wishes with "W".

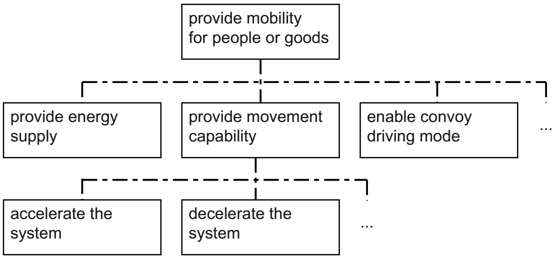

provide mobility for people or goods
provide energy supply
provide movement capability
enable convoy driving mode
...
accelerate the system
decelerate the system
...

**Fig. 4.6** Functions of the RailCab (excerpt)

**Functions:** Based on the requirements, the functions for the system under development can be defined. The aspect functions describes the hierarchical subdivision of the desired functionality of the system. A function is the general and required relationship between input and output parameters, with the aim to fulfill a task. For the specification of function hierarchies, we use a catalogue with functions which is based on the works of Birkhofer (1980) [5] and Langlotz (2000) [35]. This catalogue has been extended by functions, which describe self-optimizing functionality. Functions are realized by solution patterns and their concretizations. Starting with the overall function (e.g. provide mobility for people or goods), a subdivision into sub functions takes place (e.g. accelerate the system) until useful solution patterns can be found for the functions (e.g. linear motor). The use of solution patterns is described in a detailed manner in Sect. 4.5. Figure 4.6 shows a section of the function hierarchy of the RailCab. After the definition of the overall and sub-functions the classification scheme by Zwicky (morphological matrix) can be used, for the systematic combination of certain solutions [43]. In this classification scheme, the sub-functions and the appropriate solutions are entered into the rows of the morphological matrix. By systematically combining a solution fulfilling a specific sub-function with the solution for a neighbouring sub-function, one obtains an overall solution in the form of a possible conception. In this process, only those solution that are compatible should be combined [43].

**Active Structure:** Based on the functions and the combination of the chosen solutions the active structure for the system under development can be modeled. Thus, in contrast to the aspect environment (Black-Box view on the system and its context), the active structure concretize the system (White-Box view). The active structure defines the internal structure and the operational mode of the system. It describes system elements (e.g. chosen solutions), their attributes as well as the relationships between system elements (material, energy and information flows as well as logical relationships). Depending on the level of concretization, system elements may be described abstract (e.g. temperature sensor) or specific (e.g. resistance thermometer). If necessary, it is also possible to model elements of the environment (e.g. user) and their interaction with elements of the system (e.g. interaction of the user with the human-machine-interface of the system).

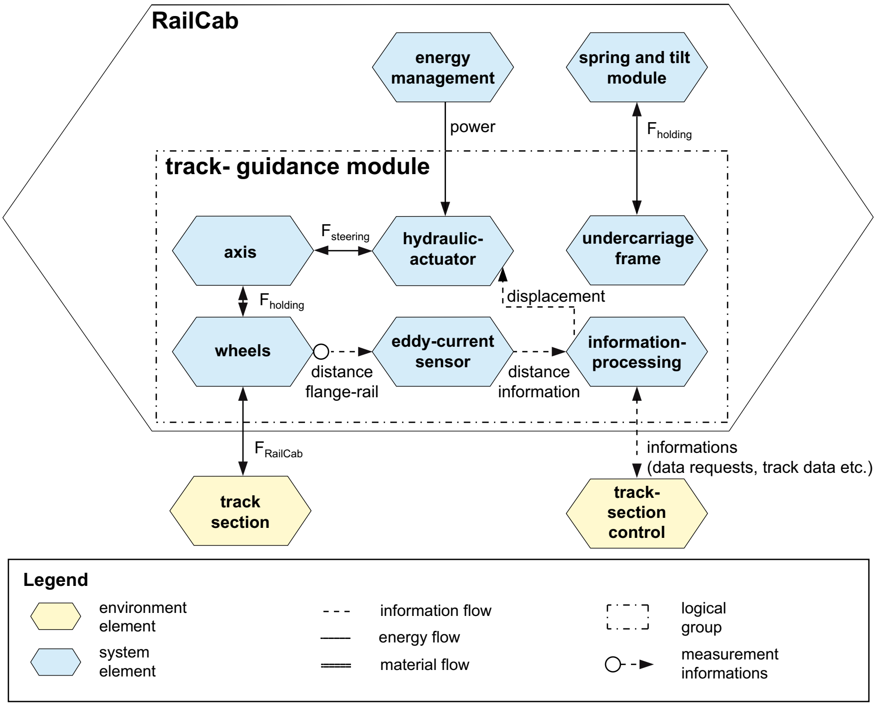
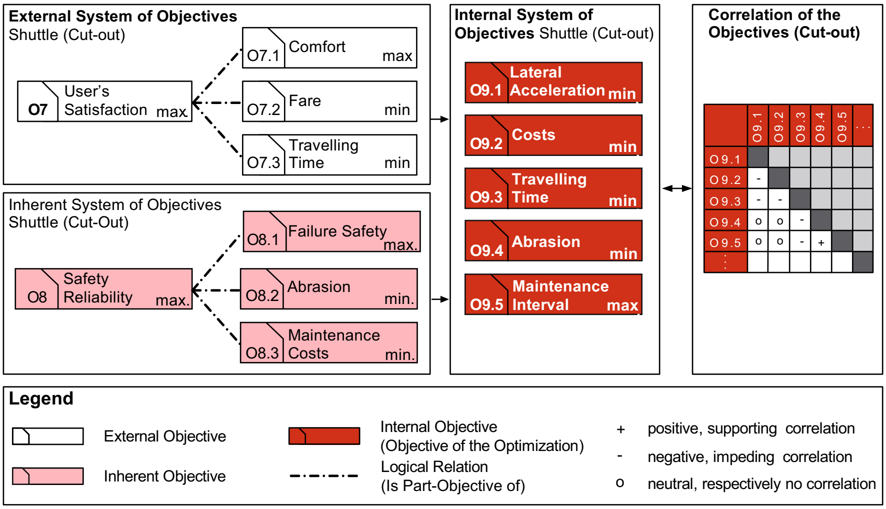


**Fig. 4.7** The partial model active structure of the RailCab (excerpt) [24]

Figure 4.7 visualizes an excerpt of the active structure for the RailCab. The active structure consists of system elements such as the Energy Management, the Spring and Tilt Module or the Track-guidance Module. To show system elements of the Track-guidance Module at the same hierarchy level as the other modules, logical groups can be used. The track-guidance module consists of eddy-current sensors (in Fig. 4.7 only one of them is shown), the hydraulic actuator, the axis, the wheels etc. The hydraulic actuator can change the position of the axis and thus of the wheels. In addition, the wheel has an mechanical contact to the rail. The eddy-current sensor measures the distance between the flange and the rail. This is specified with a measurement information flow. The information of the eddy-current sensor are sent to the information processing unit of the Track-guidance Module. The information processing unit calculates the needed displacement force, based on the sensor information and information from the track-section control. The needed displacement will be sent to the hydraulic actuator, which changes the steering position then. A closed control loop between sensor, actuator, information processing, axis, and the wheels results.

**System of Objectives:** This aspect describes external, inherent and internal objectives of the system and their interrelations. An excerpt of the system of objectives of the RailCab is shown in Fig. 4.8. External objectives are set from the outside of


External System of Objectives
Shuttle (Cut-out)
O7 User's Satisfaction max.
O7.1 Comfort max
O7.2 Fare min
O7.3 Travelling Time min
Inherent System of Objectives
Shuttle (Cut-Out)
O8 Safety Reliability max.
O8.1 Failure Safety max.
O8.2 Abrasion min.
O8.3 Maintenance Costs min.
Internal System of Objectives Shuttle (Cut-out)
O9.1 Lateral Acceleration min
O9.2 Costs min
O9.3 Travelling Time min
O9.4 Abrasion min
O9.5 Maintenance Interval max
Correlation of the Objectives (Cut-out)
Legend
External Objective
Inherent Objective
Internal Objective (Objective of the Optimization)
Logical Relation (Is Part-Objective of)
+ positive, supporting correlation
- negative, impeding correlation
o neutral, respectively no correlation

| | O9.1 | O9.2 | O9.3 | O9.4 | O9.5 | ... |
|---|---|---|---|---|---|---|
| O9.1 | ■ | | | | | |
| O9.2 | - | ■ | | | | |
| O9.3 | - | - | ■ | | | |
| O9.4 | o | o | - | ■ | | |
| O9.5 | o | o | - | + | ■ | |
| ... | | | | | | ■ |

**Fig. 4.8** The partial model system of objectives of the RailCab (excerpt) [24]

the self-optimizing system; they are set by other systems or by the user (e.g. "maximize user satisfaction"). Inherent objectives reflect the design purpose of the self-optimizing system. Inherent objectives of the RailCab are for example the objectives "maximize dependability" and "minimize energy consumption". Objectives build a hierarchy and each objective can thus be refined by sub-objectives (e.g. "maximize safety" is a possible sub-objective of the objective "maximize dependability", "maximize comfort" and "maximize driving speed" are possible sub-objectives of the objective "maximize user satisfaction"). Inherent and external objectives that are pursued by the system at a given moment during its operation are called internal objectives. The selection of internal objectives and their prioritization occurs continuously during the operation of the system. "Maximize comfort" and "maximize safety" are examples of internal objectives. Only the internal objectives are part of the self-optimization. During the operation of the self-optimizing system some of its objectives may be in conflict to each other, as they can not be pursued both to the full extend at the same time. In such cases a prioritization of the objectives has to take place. For instance, during the adjustment of the driving speed the objectives "maximize driving speed" and "minimize energy consumption" are in conflict to each other, as energy consumption typically increases with increasing driving speed. Such potential mutual influences between internal objectives are modeled in an influence matrix. In particular, the influence matrix shows which objectives may influence each other in a negative way. Such a potential negative mutual influence may be an indication for the need for self-optimization. In Sect. 4.4 we will describe how the system of objectives is designed.

**Shape:** This aspect describes the first definition of the shape of the system within the conceptual design. In particular, the working surfaces, working places and frames of the system are described in a rough manner. In mechanical engineering the aspects shape and active structure form the core of the principle solution. It is very important to model the draft of the shape during the domain-spanning conceptual design. For example, geometric restrictions for wires or mechanical components can be recognized by the different domains involved and the communication and cooperation between them is improved. Thus, expensive corrections can be avoided (e.g. too short wires in the Airbus A380 in 2006 [7]). The computer-aided modeling is performed using 3D CAD systems.

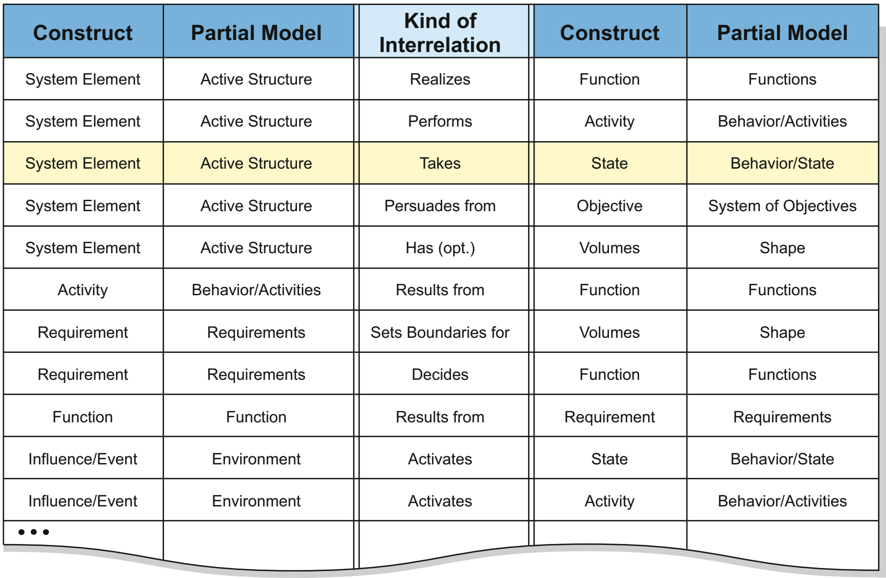
**Behavior:** A self-optimizing system is characterized by different kinds of behavior (e.g. kinematic, dynamic and reactive behavior). In order to describe the behavior of such systems, a group of behavior models is used: there are three partial models to specify the behavior. We distinguish between the partial models behavior–states, behavior–activities and behavior–sequence. The usage of the diagrams depends on the underlying development task. Additional kinds of behavior, such as kinematics, dynamics or electro-magnetic compatibility can be specified additionally.

- The partial model **behavior–states** describes all possible states of the system, all possible state transitions as well as events which initiate state transitions. Events correspond to external influences on a system or a system element as well as to already finished activities. For example, the lighting system of the RailCab can have two different states: lights on and lights off. The user-event "switch power button" causes a state transitions from "lights off" to "lights on" and vice versa.
- The partial model **behavior–activities** describes the operation process of the system , i.e. operations and tasks of the system that are performed during its operation. This especially includes operation processes which are performed in order to implement the self-optimization process (e.g. "determine the fulfillment of current system objectives", "select adequate parameters and configuration", etc.). We call such operation processes adaptation processes.
- The partial model **behavior–sequence** describes the interaction of several systems or system elements. The messages being exchanged during the interaction of those system elements are modeled in a chronological order. In Sect. 4.3 some examples of behavior description with sequence diagrams are introduced in a detailed manner.

It is necessary to alternately work on the aspects and the corresponding partial models although there is a certain order. This order is defined by the reference process for the conceptual design (cf. Sect. 3.2). In contrast to other system modeling approaches such as UML [24, 44] or SysML [20] the specification technique CONSENS is strongly interconnected with the reference process and focuses on self-optimizing mechatronic systems.

As stated before, the partial models form a coherent system and are strongly interconnected. These interconnections are modeled as **cross-references** between partial models. Tab. 4.1 shows some examples of such partial model spanning cross-references. There are e.g. bidirectional cross-references between requirements and functions, between requirements and system elements as well as between system

**Table 4.1** Interrelations between the partial models (excerpt) [24]



| Construct | Partial Model | Kind of Interrelation | Construct | Partial Model |
|---|---|---|---|---|
| System Element | Active Structure | Realizes | Function | Functions |
| System Element | Active Structure | Performs | Activity | Behavior/Activities |
| System Element | Active Structure | Takes | State | Behavior/State |
| System Element | Active Structure | Persuades from | Objective | System of Objectives |
| System Element | Active Structure | Has (opt.) | Volumes | Shape |
| Activity | Behavior/Activities | Results from | Function | Functions |
| Requirement | Requirements | Sets Boundaries for | Volumes | Shape |
| Requirement | Requirements | Decides | Function | Functions |
| Function | Function | Results from | Requirement | Requirements |
| Influence/Event | Environment | Activates | State | Behavior/State |
| Influence/Event | Environment | Activates | Activity | Behavior/Activities |
| • • • | | | | |

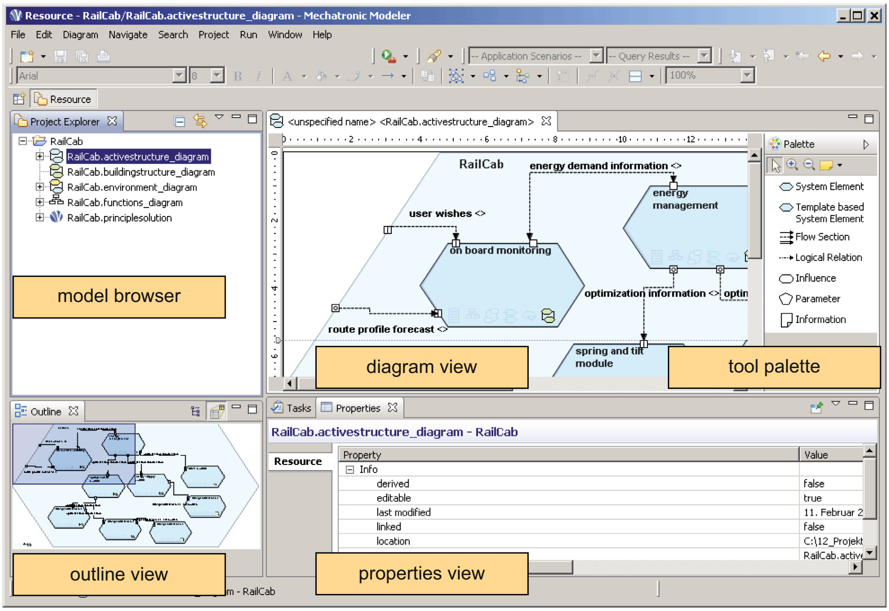
elements and functions (e.g. a "System Element" from the "Active Structure" "Realizes" a certain "Function"’ from the partial model "Functions"). Based on the specification of cross-references, analyses such as requirements traceability can be realized [23].

## 4.2 Software Support for the Specification of the Principle Solution

Rafal Dorociak and Jürgen Gausemeier

To secure the overall consistency of the principle solution and to manage its complexity, a software support is necessary. The software tool **Mechatronic Modeller** supports the creation and editing of the specification of the principle solution [23, 25]. It was developed within the research project "VireS – Virtual Synchronization of Product Development and Production System Development" founded by the German Federal Ministry of Education and Research (BMBF) in cooperation with the software company itemis. The Mechatronic Modeller is a dedicated software solution, which is fully aligned with the specification technique CONSENS. It offers a separate editor for each partial model. Figure 4.9 shows the graphical user interface of this software tool.

Within the model browser the elements of the principle solution are presented as a tree. This tree can be used to navigate within the principle solution. The currently processed partial model is shown on the right in the diagram view together with a


Resource - RailCab/RailCab.activestructure_diagram - Mechatronic Modeler
File Edit Diagram Navigate Search Project Run Window Help
-- Application Scenarios --
-- Query Results --
Arial
8
100%
Resource
Project Explorer
RailCab
RailCab.activestructure_diagram
RailCab.buildingstructure_diagram
RailCab.environment_diagram
RailCab.functions_diagram
RailCab.principlesolution
model browser
<unspecified name> <RailCab.activestructure_diagram>
RailCab
energy demand information <>
user wishes <>
energy management
on board monitoring
optimization information <>
optin
route profile forecast <>
spring and tilt module
diagram view
Palette
System Element
Template based System Element
Flow Section
Logical Relation
Influence
Parameter
Information
tool palette
Outline
outline view
Tasks
Properties
RailCab.activestructure_diagram - RailCab
Resource
properties view
m - RailCab

| Property | Value |
|---|---|
| Info | |
| derived | false |
| editable | true |
| last modified | 11. Februar 2 |
| linked | false |
| location | C:\12_Projekt |
| | RailCab.active |

**Fig. 4.9** Screenshot of the Mechatronic Modeller showing the active structure editor [23]

tool palette. Within this diagram view the respective partial model can be modified. Using the tool palette new elements can be added. The outline view (bottom left) continuously shows the outline of the whole diagram. Within this view the user can navigate through the whole diagram. This allows the user to navigate to sections of the partial model which are currently not shown in the diagram view.

A so-called metamodel has been defined for the specification technique. It defines [52]:

- which model elements are available during the description of the principle solution as well as how they are related to each other (abstract syntax); for instance, states can be linked to other states using a relation, and
- criteria for well-formedness (static semantics); for instance, the names of states must be unique in the scope of the statechart.

In particular, the metamodel describes all possible interrelations between the different partial models.

The Mechatronic Modeller is based upon this metamodel. Thus, each principle solution modeled with the Mechatronic Modeller is computer-internally represented as a data model, which is an instance of this metamodel. Given that all constraints have been formally defined in the metamodel, the conformance of such a model to the metamodel can be easily checked, allowing immediate feedback for the developer in case of modeling errors.

In addition to the metamodel, the following aspects of the specification technique had to be defined during the development of the tool:

- how the models will be graphically represented (concrete syntax); for instance, a state is represented as a rounded rectangle, and
- the meaning of the different modeling constructs in a particular principle solution (dynamic semantics); for instance, state transitions are triggered when the event at the transition is fired.

Using a precise definition of the dynamic semantics of a language, a model can be simulated, analyzed, and/or formally verified [23, 25].

The Mechatronic Modeller addresses all particularities of the specification technique. The very important aspect of usability can therefore be appropriately addressed. The aim is to hide the complexity of the model from the developer. Thus, several functions have been incorporated into the Mechatronic Modeller which support working with the specification technique and make the tool more comfortable and enables an intuitive use. In particular, complex manipulations such as partial model reorganization by incorporating or deleting of hierarchy levels, are provided by the tool. Furthermore, cross-references between elements of different partial models are stored in the data model. Thus, Mechatronic Modeller is capable of handling complex dependencies between elements of different partial models within the principle solution. For example, it is possible to check which requirements have not yet been realized by functions or system elements (static semantics checks). In particular, requirements traceability is possible, e.g. if a particular system element needs to be exchanged, then the developer can examine which requirements had to be originally met by it [23].

## 4.3 Consistency Analysis of Application Scenarios

Christian Brenner and Wilhelm Schäfer

During the development of a self-optimizing system, the definition of the system behavior is highly important. Application scenarios usually form the earliest description of the system behavior (cf. Sect. 4.1). In the course of the progressing conceptual design this description is further refined; the partial models behavior–states and behavior–activities are used for this purpose. Eventually, these partial descriptions of the behavior are combined into one scenario-spanning model of the overall system behavior. It is a challenge to ensure the consistency of the aforementioned partial descriptions of behavior. In order to support this difficult task, a method to ensure the consistency of application scenarios has been developed. This method is based on a formal description of application scenarios [29]. It allows for the early detection and correction of inconsistencies between partial descriptions of the behavior.

Figure 4.10 shows the procedure model of the method. The starting point is the definition of the discrete system behavior by using application scenarios. The method consists of the following three phases:

**Phase 1 - formalization of the discrete behavior:** In order to automatically detect inconsistencies in the modeled system behavior, the models of interest have to

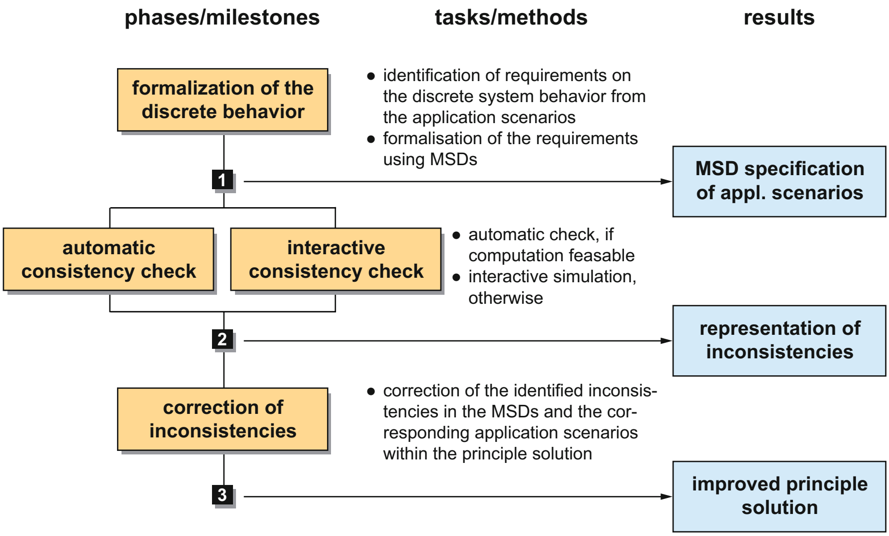

phases/milestones
tasks/methods
results
formalization of the discrete behavior
• identification of requirements on the discrete system behavior from the application scenarios
• formalisation of the requirements using MSDs
1
MSD specification of appl. scenarios
automatic consistency check
interactive consistency check
• automatic check, if computation feasable
• interactive simulation, otherwise
2
representation of inconsistencies
correction of inconsistencies
• correction of the identified inconsistencies in the MSDs and the corresponding application scenarios within the principle solution
3
improved principle solution

**Fig. 4.10** The procedure model of the method for ensuring the consistency of application scenarios [29]

be specified in a formal way. We will demonstrate this by using the example of the application scenario "AS12: Drive onto next track section" shown in Fig. 4.4 (Sect. 4.1, p. 122). It defines how the RailCab and the track section control interact, when the RailCab is about to enter a track section. In particular, it defines requirements regarding the system behavior using text (e.g. in the principle solution of AS12: "[...] the RailCab, when reaching the end of the track section, sends a request [...]") and illustrations. Also, it may contain assumptions about the system environment (e.g. in the description of AS12: "[the RailCab] is at some point notified that it approaches the end of the track section."). However, these requirements and assumptions are at first specified only informally using natural language. In order to process the application scenario automatically, we first need to formalize them. We use **Modal Sequence Diagrams** (MSDs) for this purpose [30]. They have been adapted for mechatronic systems [29] by taking into account real-time behavior and assumptions about the system environment. We distinguish requirement MSDs and assumption MSDs. Requirement MSDs model requirements on the system based on the respective application scenario. Assumption MSDs specify assumptions about the environment of the system.

Figure 4.11 shows the MSD specification formalizing the application scenario "AS12" from Fig. 4.4. It consists of three MSDs. The topmost MSD and the one in the middle are both requirement MSDs. The bottom MSD is an assumption MSD, which is indicated by the label *«EnvironmentAssumption»*.

In each MSD, the vertical dashed lines, called *lifelines*, represent the participants of the respective scenario. At the top of each lifeline, a label in a hexagon defines the corresponding system element or environment element. The application

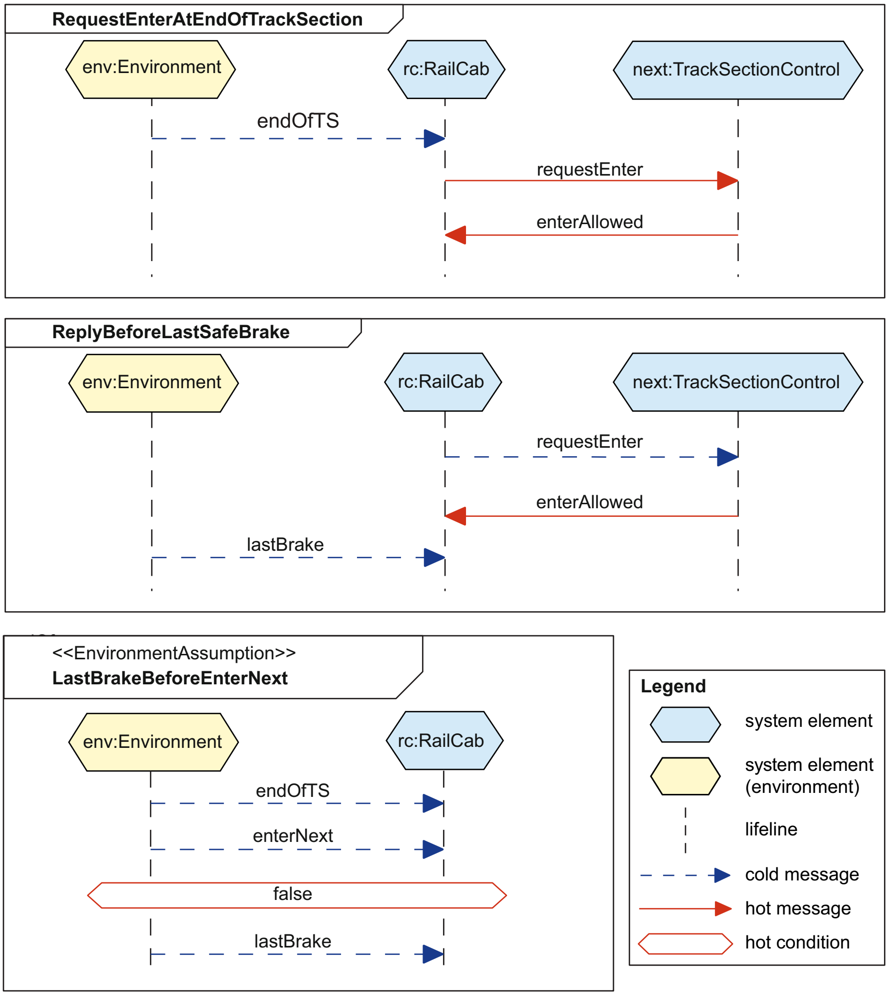


**Fig. 4.11** Modal Sequence Diagrams for the application scenario of Fig. 4.4 [29]

scenario in Fig. 4.4 explicitly mentions a RailCab and a section control of the next track section. Consequently, both requirement MSDs contain a system element "rc" (representing a RailCab) and a system element "next" (representing the upcoming track section control). In addition, the environment is represented by the lifeline "env". The horizontal arrows in the MSDs are messages that are exchanged between system elements. Each arrow starts at the lifeline of the sender and ends at the lifeline of the receiver. The label at the arrow defines the type of message that is exchanged. Dashed arrows represent messages that may occur, but do not have to (e.g. *endOfTS* in the topmost MSD). These are called *cold messages*. Solid arrows represent messages that are required to occur (e.g. *requestEnter* in the topmost MSD). We refer to them as *hot messages*. The vertical position of the arrows in the MSD defines the chronological order of the corresponding messages: the topmost message is expected to occur first, then the one below it, and so on.

As mentioned before, the MSDs in Fig. 4.11 formalize the requirements and assumptions that are informally expressed in the description of the application scenario in Fig. 4.4. For example, in the topmost MSD, the cold message *endOfTS* models the notification by the environment mentioned in the first sentence of the description of the development task of AS12 ("[...] the RailCab [...] is at some point notified that it approaches the end of the track section"). The system has to realize the requirements specified by the hot messages *requestEnter* and *enterAllowed* only after receiving the message *endOfTS*. According to the MSD, both messages have to be sent in this particular order. They model the requirements stated in the first sentence in the principle solution part of the description of the AS12 in Fig. 4.4 ("[...] the RailCab, when reaching the end of the track section, sends a request to enter the next track section to the section control [...]", "Then the section control replies [...]"). The second MSD formalizes the remaining requirement in a similar way: the reply must be sent on time, i.e. as long as braking is still allowed.

When the first message of an MSD is exchanged, this MSD becomes *active*. For example, the topmost MSD becomes active when the environment sends the message *endOfTS* to the RailCab "rc". Then, the message *requestEnter* is expected to occur next. The next expected message of an active MSD is referred to as an *enabled message*. After an enabled message has been sent, the next enabled message is the subsequent message in the order defined by the MSD. If, for example, the message *requestEnter* in the topmost MSD is enabled and is actually sent, *enterAllowed* is the next enabled message.

If a message occurs that is included in an active MSD, but is not currently enabled, then this is a *violation* of the MSD. Messages that are not included in an MSD can never violate it (e.g. *lastBrake* can never violate the topmost MSD). A violation of an MSD is a *cold violation*, if the enabled message is a cold message. It is a *hot violation*, if the enabled message is hot. A cold violation of an MSD turns the MSD inactive again. Assume, for example, that the MSD at the bottom of Fig. 4.11 is active and the message *enterNext* is enabled. Then, the message *lastBrake*, if it is sent, causes a cold violation and turns the MSD inactive. Note, that a cold violation does not indicate incorrect behavior. It only means that there is an allowed deviation from the scenario modeled by the MSD. A hot violation, on the contrary, models forbidden behavior of the system or unexpected behavior of the environment. If, for instance, the topmost MSD is active because *endOfTS* was sent, the next enabled message is the hot message *requestEnter*. If *requestEnter* is never sent, or if *enterAllowed* or *endOfTS* is sent instead, a hot violation occurs. Unlike a cold violation, a hot violation of a requirement MSD may never occur in the real system. A hot violation of an assumption MSD means that the environment does not behave as assumed. The system may not be used in such an environment.

After modeling the requirement MSDs and assumption MSDs for all application scenarios, they are combined into one complete MSD specification for the whole system. In Phase 2 this MSD specification is analyzed for inconsistencies.

**Phase 2b - interactive consistency check:** The *automatic consistency check* is computationally very expensive and, hence, not applicable for very complex MSD specifications. In case of such complex specifications, the engineer can instead perform an *interactive simulation*. This simulation does a stepwise evaluation of the specified behavior. The simulation can be influenced by the engineer by selecting the actions of the system, if the specification allows several alternatives. The engineer performing the simulation can also influence the simulated behavior of the environment. However, the interactive simulation only allows to consider individual simulation runs. For large systems, it is usually not possible to cover all possible executions. Therefore, the interactive simulation alone can not prove that the discrete behavior of the system described with application scenarios is completely consistent.

**Phase 3 - correction of inconsistencies:** Here the inconsistencies found during Phase 2, if any, are corrected. This happens in two ways. On the one hand, contradicting requirements are directly corrected by modification of the respective requirement MSDs. On the other hand, contradictions in the requirements can often be resolved by modeling additional environment assumptions using assumption MSDs. In both cases, the engineers also adapt the textual descriptions of the application scenarios in accordance with the changes in the MSD specification.

**Tool support:** For our method a software tool, called Scenario Tools, has been developed. It allows the engineer to create and edit MSD specifications as well as to validate the specification by using either the automatic consistency check or the interactive simulation, as described above.

All in all, with the presented method and tool the overall consistency of the behavior described with application scenarios is improved. This is very important, as potential inconsistencies in the behavior specification would otherwise be difficult to detect in later product development phases and could lead to problems regarding reliability, safety or availability.

## 4.4 Design of the System of Objectives

Rafal Dorociak and Jürgen Gausemeier

Sections 4.4 - 4.5 introduce methods, which support the development of the principle solution. We will begin with the specification of the system of objectives. The partial model **system of objectives** describes the objectives of the system which are subject to self-optimization and are therefore of particular importance for the design of self-optimizing systems. For the specification of the system of objectives, a method has been developed by Pook (2011) [45]. Using the method objectives of the system, their relationships to each other and potential conflicts are identified based on the description of the principle solution for the system. Altogether the method assists developers with the design of the information processing of the system, which then realizes self-optimization. During the development of the method some ideas and concepts from the Fault Tree Analysis (FTA) [4, 6, 32] and Failure Mode and

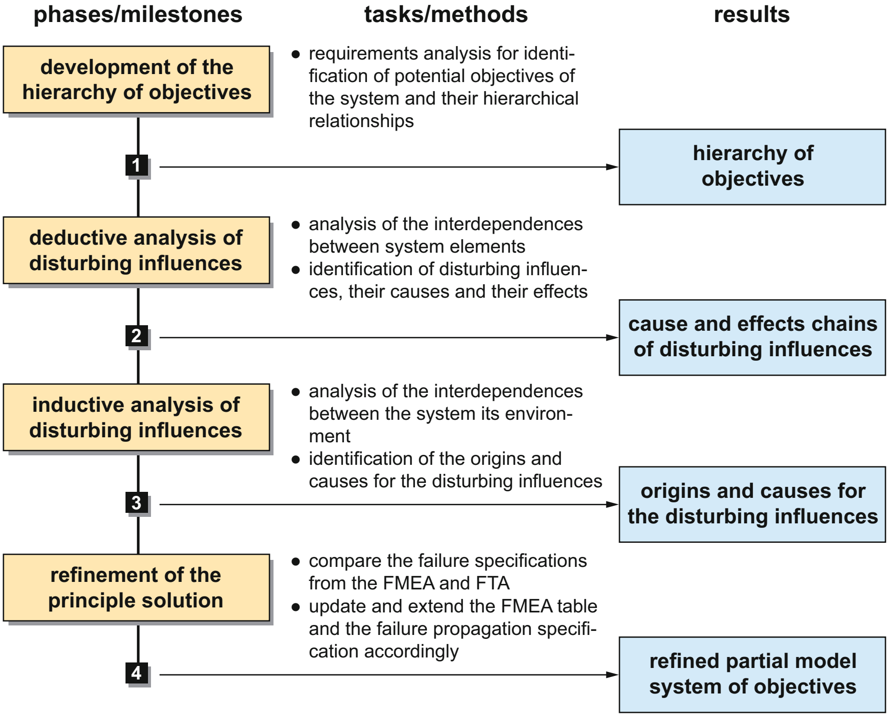
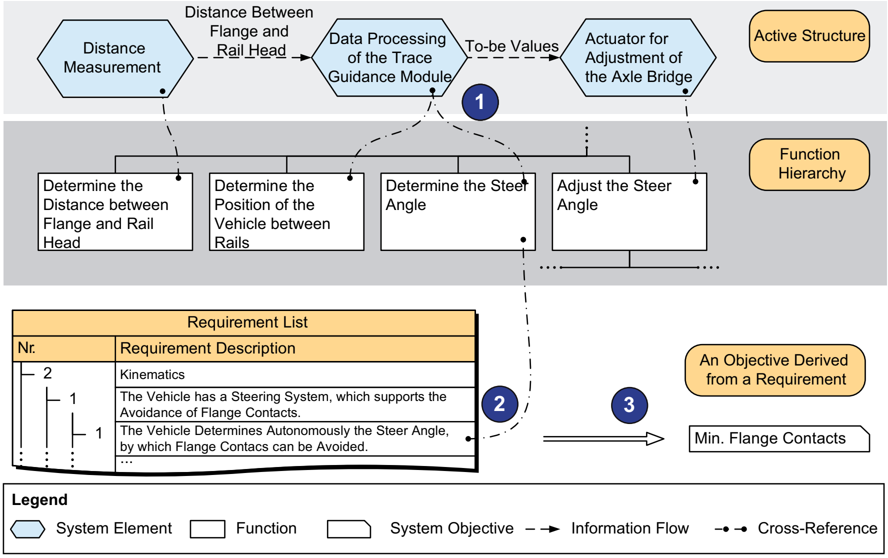

phases/milestones
tasks/methods
results
development of the hierarchy of objectives
• requirements analysis for identification of potential objectives of the system and their hierarchical relationships
1
hierarchy of objectives
deductive analysis of disturbing influences
• analysis of the interdependences between system elements
• identification of disturbing influences, their causes and their effects
2
cause and effects chains of disturbing influences
inductive analysis of disturbing influences
• analysis of the interdependences between the system its environment
• identification of the origins and causes for the disturbing influences
3
origins and causes for the disturbing influences
refinement of the principle solution
• compare the failure specifications from the FMEA and FTA
• update and extend the FMEA table and the failure propagation specification accordingly
4
refined partial model system of objectives

**Fig. 4.12** The procedure model of the method for the design of the system of objectives

Effects Analysis (FMEA) [4, 6, 31] were adapted. Figure 4.12 shows the constituent phases of the method and the corresponding milestones:

**Phase 1 - development of the hierarchy of objectives:** The starting point is the identification of the possible objectives of the system. The objectives are contained in the list of requirements. The list of requirements of a complex mechatronic system usually contains a great number of entries, e.g. up to several thousands of requirements. In order to extract the objectives of the system from the list of requirements, the cross-references between the partial models of the principle solution are used (e.g. cross-references between functions and functional requirements and between system elements and functions). The starting points are the system elements which realize the information processing. The hierarchical dependencies between the identified objectives are found, analogously. The first phase is performed only once. The result is the hierarchy of the objectives of the system. Figure 4.13 shows a cut-out of the principle solution for the trace guidance module of the RailCab system. System elements, which realize the information processing, are identified first, e.g. the system element "data processing of the trace guidance module" (1). The corresponding requirements are then traced using the respective cross-references. One requirement corresponding to the system element "data processing of the trace guidance module" is the requirement "2.1.1. The vehicle determines autonomously


Distance Measurement
Distance Between Flange and Rail Head
Data Processing of the Trace Guidance Module
To-be Values
Actuator for Adjustment of the Axle Bridge
Active Structure
1
Function Hierarchy
Determine the Distance between Flange and Rail Head
Determine the Position of the Vehicle between Rails
Determine the Steer Angle
Adjust the Steer Angle
2
3
An Objective Derived from a Requirement
Min. Flange Contacts
Legend
System Element
Function
System Objective
Information Flow
Cross-Reference

| Requirement List | |
|---|---|
| Nr. | Requirement Description |
| 2 | Kinematics |
| 1 | The Vehicle has a Steering System, which supports the Avoidance of Flange Contacts. |
| 1 | The Vehicle Determines Autonomously the Steer Angle, by which Flange Contacs can be Avoided. |
| | ... |

**Fig. 4.13** Development of the hierarchy of objectives of the RailCab System (cut-out) [45]

the steering angle, where flange contacts can be avoided" (2). From this requirement the system objective "minimize flange contacts" is derived (3).

Further requirements and objectives are found, analogously. For instance, the requirement ("13.1. The vehicle has facilities for increasing the comfort during the transport of people"). From this requirement the objective "maximize traveling comfort" is derived. In the context of the RailCab system the objective "maximize traveling comfort" is subordinated to the previously identified objective "minimize flange contacts". In the partial model system of objectives this fact is modeled using the "is part of" relationship; a hierarchy of objective is constructed in this manner.

**Phase 2 - deductive analysis of disturbing influences:** Each of the objectives of the system found in Phase 1 is further examined. The disturbing influences are identified, which may occur during the operation of the system and have negative influence on the objective of the examined system. In order to identify these disturbing influences a Fault Tree [4] is built for each objective of the system. The top event of the Fault Tree is first postulated. It usually has the form: "the system is being disturbed while pursuing the objective x". The deductive analysis follows. It can be thought of as a "how-can" analysis [15]. The developer works in a top-down manner to find specific combinations of events, which could have occurred for the top-event to have taken place. We propose the following schema: The left branch of the tree describes cases, in which the system is disturbed by provision of premises for the pursuit of the objective under consideration; the inputs of the system element are examined here. The right branch describes cases, in which the premises are given, but the system is disturbed during the execution of activities for the pursuit of the

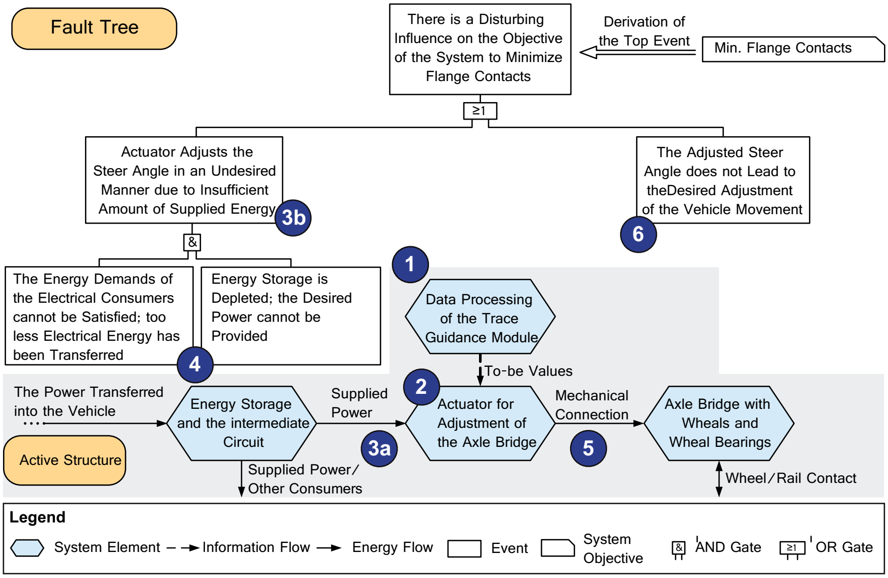

Fault Tree
There is a Disturbing Influence on the Objective of the System to Minimize Flange Contacts
Derivation of the Top Event
Min. Flange Contacts
≥1
Actuator Adjusts the Steer Angle in an Undesired Manner due to Insufficient Amount of Supplied Energy
3b
The Adjusted Steer Angle does not Lead to theDesired Adjustment of the Vehicle Movement
6
&
The Energy Demands of the Electrical Consumers cannot be Satisfied; too less Electrical Energy has been Transferred
Energy Storage is Depleted; the Desired Power cannot be Provided
4
1
Data Processing of the Trace Guidance Module
To-be Values
The Power Transferred into the Vehicle
...
Energy Storage and the intermediate Circuit
Supplied Power
2
Actuator for Adjustment of the Axle Bridge
Mechanical Connection
Axle Bridge with Wheals and Wheal Bearings
3a
5
Active Structure
Supplied Power/ Other Consumers
Wheel/Rail Contact
Legend
System Element
Information Flow
Energy Flow
Event
System Objective
AND Gate
OR Gate

**Fig. 4.14** Deductive analysis of disturbing influences of the RailCab system (cut-out) [45]

objective; here, mainly the outputs of the system element under consideration are examined.

Let us consider the objective "minimize flange contacts". The respective cut-out of the active structure for the RailCab system and the corresponding Fault Tree are shown in Fig. 4.14. From Phase 1 we know, that the objective originated from the system element "data processing of the trace guidance module" (Fig. 4.14 (1)). The "desired values" are sent to the system element "actuator for adjustment of the axle bridge" (2). We construct the left branch of the tree by following the aforementioned schema. The system element "actuator for adjustment of the axle bridge" receives power from the "energy storage and the intermediate circuit" (3a). Thus the event "actuator adjusts the steer angle in an undesired manner due to insufficient amount of supplied energy" (3b) is integrated in the left branch of the Fault Tree. This event is further refined in cooperation with the respective developers. In the course of this, two subordinated events are incorporated into the Fault Tree (4). We now proceed with the right branch of the tree. One of the outputs of the "actuator for adjustment of the axle bridge" is the flow "mechanical connection" (5). It is examined and the event "The adjusted steer angle does not lead to the desired adjustment of the vehicle movement" is integrated into the right branch of the Fault Tree (6). The newly incorporated event is then further refined and the procedure continues.

**Phase 3 - inductive analysis of disturbing influences:** Usually not all of the disturbing influences are found in Phase 2. Therefore an inductive analysis similar to the FMEA is performed as well. An inductive analysis can be thought of as a "what-if" analysis [15]. The developer asks: What if this system element failed, what are the consequences, what are the possible causes etc.? The developer starts

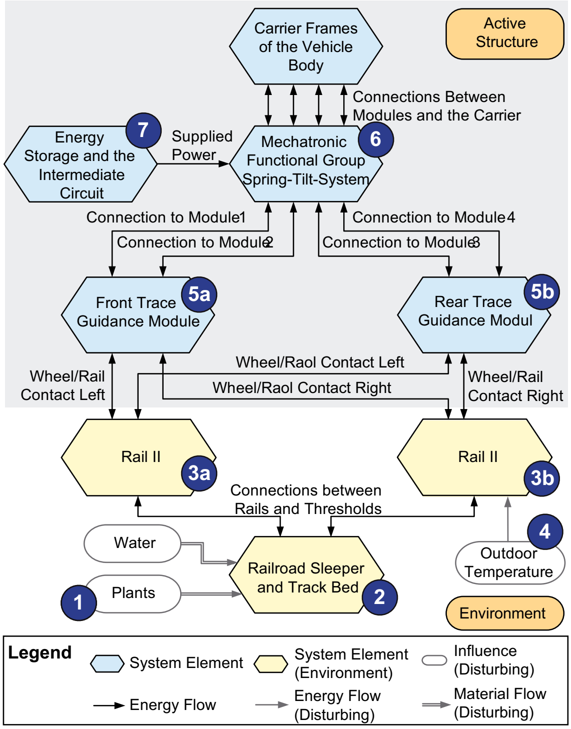


**Fig. 4.15** Inductive analysis of disturbing influences of the RailCab (cut-out) [45]

with the influences from the environment of the system and investigates how they propagate through the active structure of the system. Fault Trees from Phase 2 are extended with regard to the newly gathered information.

Figure 4.15 shows a cut-out of the partial models environment and active structure of the RailCab. The track bed changes due to growing plants (Fig. 4.15 (1)). As a result the position of the sleeper in the track changes (2), as well as the position of the rails (3a and 3b). In combination with environmental influences (4) deformations of the rail occur. These lead to movements of the vehicle body (5a and 5b) which are transferred to the Active Suspension Module through mechanical connections (6). In order to dampen vibrations and tilt the vehicle body during curves, more energy has to be provided to the active suspension (7). The increased energy demands of the Active Suspension Module can eventually lead to the depletion of the stored energy. The newly gathered information is incorporated into the Fault Trees, which have been constructed in Phase 2.

**Phase 4 - refinement of the principle solution:** The Fault Trees are evaluated and the principle solution is extended accordingly. In particular, the conflicts between objectives of the system are identified. Objectives of the system which do not exhibit any conflicts with other objectives are marked as not relevant for the self-optimization and removed from the partial model system of objectives. The description of the identified conflicts is incorporated into the system of objectives.

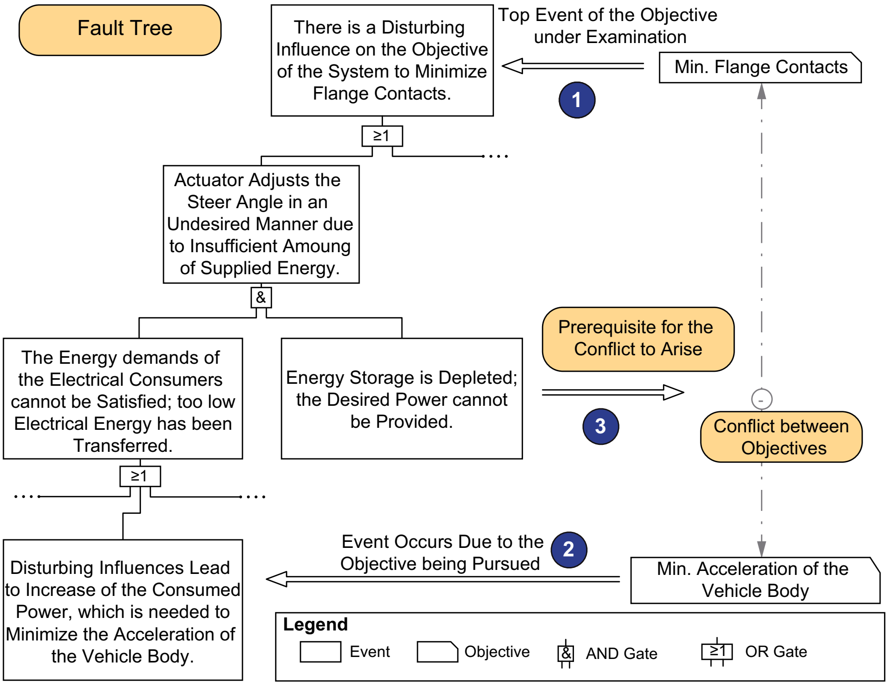

Fault Tree
There is a Disturbing Influence on the Objective of the System to Minimize Flange Contacts.
Top Event of the Objective under Examination
Min. Flange Contacts
1
≥1
Actuator Adjusts the Steer Angle in an Undesired Manner due to Insufficient Amoung of Supplied Energy.
&
The Energy demands of the Electrical Consumers cannot be Satisfied; too low Electrical Energy has been Transferred.
Energy Storage is Depleted; the Desired Power cannot be Provided.
Prerequisite for the Conflict to Arise
3
-
Conflict between Objectives
≥1
Disturbing Influences Lead to Increase of the Consumed Power, which is needed to Minimize the Acceleration of the Vehicle Body.
Event Occurs Due to the Objective being Pursued
2
Min. Acceleration of the Vehicle Body
Legend
Event
Objective
& AND Gate
≥1 OR Gate

**Fig. 4.16** Identification of conflicts between objectives based on the Fault Tree (cut-out) [45]

The result of this phase is the extended system of objectives, showing all objectives relevant for the self-optimization and the possible conflicts between them.

A cut-out of the refined Fault Tree for the RailCab is shown in Fig. 4.16. The event "There is a disturbing influence on the objective of the system to minimize flange contacts" (1) was derived from the objective "minimize flange contacts" in Phase 2. Starting with this event the Fault Tree is further examined. The event "Disturbing influences lead to increase of the consumed power, which is needed to minimize the acceleration of the vehicle body" (2) is found. It occurs if the objective "minimize acceleration of the vehicle body" is being pursued. According to the Fault Tree there is a potential conflict between both objectives. The prerequisite for the occurrence of conflict is also derived from the Fault Tree (3). Such information is very relevant for the further realization of the self-optimization process.

Phases 2 - 4 are conducted for each objective identified in Phase 1. Possible conflicts as well as prerequisites for their occurrence are recorded in the partial model system of objectives (for an example see Fig. 4.8 in Sect. 4.1). In particular, objectives, which are in conflict with the objective "maximize reliability", and the respective prerequisites are identified. The gathered information forms a basis for the further improvement and extension of the principle solution. In particular, new measuring system elements and corresponding information flows have to be incorporated into the active structure. Furthermore, activities for gathering information about disturbing influences and conflicts between objectives as well as for recognizing and mitigating these are integrated into the partial model behavior–activities. In particular, condition monitoring and performance assessment are implemented [37, 51]. Altogether, the system under development is made more reliable. The process of the improvement of the principle solution can be supported by solution patterns, which we will explain in Sect. 4.5.

## 4.5 Design Framework for the Integration of Cognitive Functions Based on Solution Patterns

Harald Anacker, Roman Dumitrescu, and Jürgen Gausemeier

The following approach by Dumitrescu (2011) defines a design framework for the development of cognitive functions based on solution pattern during early design phases – conceptual design or system design. By using this framework developers can systematically integrate those functions within the principle solution of a self-optimizing system. The result is the early specification of the information processing within the architecture of the Operator-Controller-Module. Later on, during the concretization, this enables the final implementation of the cognitive functions [10]. The basis for this approach are the research fields of mechatronics, that cover the technical demand, and cognitive science, from which many results about intelligent behavior and structures have to be considered. The design framework itself covers four basic steps, which will be introduced in the following four sections.

The core of the framework is a procedure model (4.17). It gives an overview of the steps that have to be carried out during the conceptual design to integrate cognitive functions in the principle solution and the concrete results of the correlative steps.

Moreover, the procedure model defines what methods or tools should be used during which step. The procedure model connects all the parts of the design systematics in a logical sequence for their application. The different phases will be explained in the following subsections.

### *4.5.1 Systems Analysis*

The objective of the first phase is to create a statement if the significant improvement of system performance can be expected through the integration of cognitive functions. Consequently, this phase clarifies the requirements for using cognition and its respective methods. In order to detect the potential use of cognitive functions, the *method for objective function analysis* is used. With this method, the necessity of using active paradigms for self-optimization can be established. This is divided into four successive steps:

1. **Identification of relevant influential specifications:** In the first phase, if not already done, the objectives of the system are identified (e.g. low energy

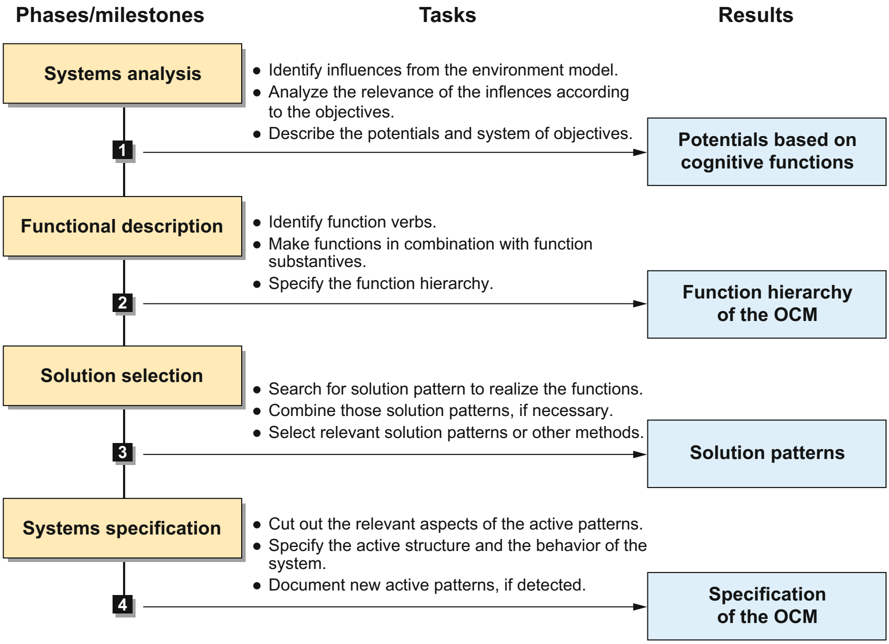
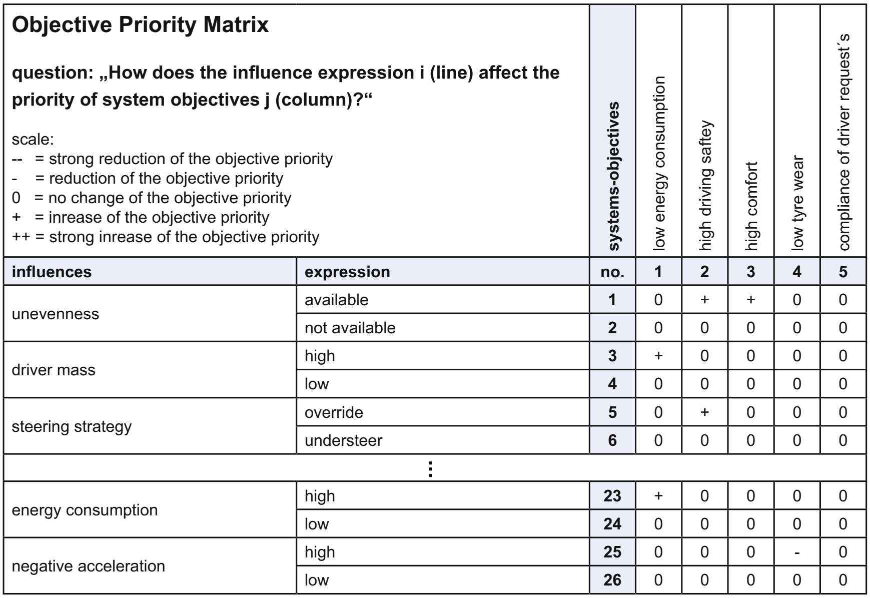

Phases/milestones
Tasks
Results
Systems analysis
• Identify influences from the environment model.
• Analyze the relevance of the inflences according to the objectives.
• Describe the potentials and system of objectives.
1
Potentials based on cognitive functions
Functional description
• Identify function verbs.
• Make functions in combination with function substantives.
• Specify the function hierarchy.
2
Function hierarchy of the OCM
Solution selection
• Search for solution pattern to realize the functions.
• Combine those solution patterns, if necessary.
• Select relevant solution patterns or other methods.
3
Solution patterns
Systems specification
• Cut out the relevant aspects of the active patterns.
• Specify the active structure and the behavior of the system.
• Document new active patterns, if detected.
4
Specification of the OCM

**Fig. 4.17** Procedure model for the specification of the OCM with cognitive functions [10]

consumption or the level of safety of a vehicle.) In addition, a context analysis has to determine the influences that in principle the objectives can influence. It is important that potential forms of the influences are determined.

2. **Illustration of the effect of the influential specifications:** The next step, the influential specifications and the objectives of the system are placed in relationship to each other. It is interesting to note how the influential specifications affect the objective priority. The objective priority characterizes the importance of a objective under the given influence. Accordingly, an increase in the objective priority should be accompanied by an increase in the objective weighing. Here the **Objective Priority Matrix** lists the influence like its specifications in rows and in the columns the rows. The matrix can then change the priority objective due to the influential specifications (strong decrease, decrease, no effect, increase, and strong increase). 4.17 shows a portion of the objective priority matrix of the flexible road vehicle Chameleon 2.3.
3. **Educational relevant situations:** In the third phase non-relevant objectives and influential specifications are stricken from the Objective Priority Matrix of operation. It is evident that a column (irrelevant objective) or a row (irrelevant influential specification) is evaluated neutrally with a "0". Afterwards situations are formed from the leftover influential specifications. Situations are consistent combinations of influences. Combinations of influential specifications that cannot happen in reality should be excluded.



**Objective Priority Matrix**

**question: „How does the influence expression i (line) affect the priority of system objectives j (column)?“**

scale:
-- = strong reduction of the objective priority
- = reduction of the objective priority
0 = no change of the objective priority
+ = inrease of the objective priority
++ = strong inrease of the objective priority

| | | systems-objectives | low energy consumption | high driving saftey | high comfort | low tyre wear | compliance of driver request´s |
|---|---|---|---|---|---|---|---|
| **influences** | **expression** | **no.** | **1** | **2** | **3** | **4** | **5** |
| unevenness | available | **1** | 0 | + | + | 0 | 0 |
| | not available | **2** | 0 | 0 | 0 | 0 | 0 |
| driver mass | high | **3** | + | 0 | 0 | 0 | 0 |
| | low | **4** | 0 | 0 | 0 | 0 | 0 |
| steering strategy | override | **5** | 0 | + | 0 | 0 | 0 |
| | understeer | **6** | 0 | 0 | 0 | 0 | 0 |
| ⋮ | | | | | | | |
| energy consumption | high | **23** | + | 0 | 0 | 0 | 0 |
| | low | **24** | 0 | 0 | 0 | 0 | 0 |
| negative acceleration | high | **25** | 0 | 0 | 0 | - | 0 |
| | low | **26** | 0 | 0 | 0 | 0 | 0 |

**Fig. 4.18** Objective Priority Matrix (example: demonstrator Chameleon) [10]

4. **Evaluation of the situation-dependency of the objective:** In the final step, the objective priorities are counted from the Objective Priority Matrix for each situation and a situation-dependent priority of each line determined. The outcome of this is the degree of dependence of the objective. Thus, an objective of single priority in a given situation can stand on its own or distribute itself to one or many other objectives. If the objectives of situations across other situations prioritize differently, then this is the first criteria for the integration of cognitive functions. Furthermore, the proportion of shared objectives is of importance. With a high proportion of shared objectives this suggests that this objective is closely related to other objectives and its weight should not be from the outset determined and isolated by the developer.

## *4.5.2 Functional Description*

In order to realize a self-optimization process, optimization systems have to perform information processing functions such as to communicate, to share knowledge or to extract information. These functions are known as **cognitive functions**. Even though there is no common accepted definition of cognition, there is a common sense that cognition intervenes between the perception and the behavior of a system in the way, that certain stimuli does not always result in the same reaction. Therefore, cognition can be characterized as the ability that does not only enable autonomy and adaptability, but also a more reliable, effective ,and viable system with regard

**Table 4.2** Examples of basic information processing functions [10]

| basic functions | specific functions for concretization |
|---|---|
| to acquire | to (call, update, ask, receive, measure) |
| to process | to (prepare, contain, divide, compensate, choose, filter, convert, delete, save, compare, evaluate) |
| to transfer | to (command, deactivate, activate, provide, set, inform, send, allocate, transmit) |

**Table 4.3** Examples of cognitive functions [10]

| complex functions (abstract) | declaration - exemplary combination of basic functions |
|---|---|
| to analyze | A system element receives information or makes an explicit request. In addition, the information is analyzed based on existing and additionally requested information. In conclusion, the results are transmitted to one or more system elements. to call - to compare - to transmit |
| to classify | A system element receives information or makes an explicit request. The element compares the information with already existing information. Depending on the comparison; the information gets a new evaluation and classified. This classification is saved and transmitted to other system elements. to receive - to compare/prepare/save - to allocate |

to its purpose. Strube (1996) distinguishes the following cognitive functions on a psychological level [54]: to observe, to recognize, to encode, to store, to remember, to think, to solve problems, to control motor function and to use language. Thus, cognitive functions are basically information processing functions which not only formalize new information, but also connect new information with existing internal information. Since cognitive functions process information – and this is the main assumption of cognitive science – they are calculation processes and can therefore also be implemented in technical systems.

To name the functions of technical systems different noun-verb-catalogs for mechanical engineering have been developed [34, 35, 43, 54]. Due to an outstanding overall catalog of information processing functions, we have developed one as the first step to integrate software specific aspects in the principle solution. According to the IPO-Model (Input-Processing-Output), we have distinguished between three basic functions: the functions of acquiring, processing and transferring information. In respect to several other functions were identified to concretize those basic functions. For example, the acquisition of information can be done in a passive (e.g. to receive information) or an active way (e.g. to retrieve information). All in all, 24 functions have been documented Table 4.2. Furthermore, other types of functions have occurred: functions, which were a combination of the basic functions (e.g. to analyze or to classify). These functions are named complex cognitive functions of information processing Table 4.3.

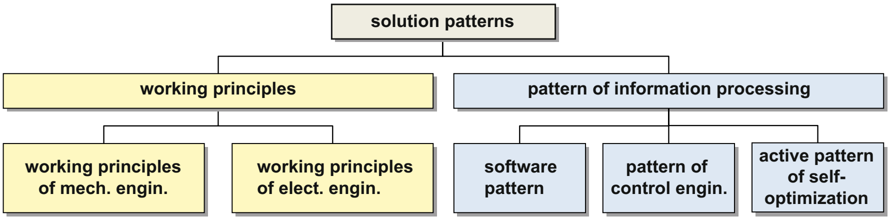

solution patterns
working principles
pattern of information processing
working principles of mech. engin.
working principles of elect. engin.
software pattern
pattern of control engin.
active pattern of self-optimization

**Fig. 4.19** Classification of solution patterns [10]

### *4.5.3 Solution Selection*

In this phase, the potential solutions for the functions are identified. In order to achieve this, the sub-functions must be allocated at the lowest level of the function hierarchy of partial solutions and successively fill the higher-level functions up until the complete function. The prerequisite for finding solutions for the integration of cognitive functions is an adequate representation. For this purpose, recurring solutions in the form of **solution patterns** are prepared.

In the architecture the idea was formulated in this context that the core of a solution can be described as a pattern for a specific problem that can be drawn upon in this analogous problem situations [1]; recurring problems are not to be solved from the ground up every time. This is valid in an analogous way in mechanical and electrical engineering as well as in control and software engineering and also for the conception of intelligent behavior of self-optimizing systems. In this respect, the structuring of the solution patterns during the development processes presents an important foundation for the development of these systems.

Generally a pattern describes a recurrent problem in a definite context and the core for this problem, i.e. the structure and behavior of the characteristic elements of possible solutions in a generalized form. Based on this assumption, 4.19 presents a depicted classification of the solution patterns.

We differentiate solution patterns that contact physical effects and patterns that serve exclusively for processing information. The construction doctrine of mechanical and electrical engineering identifies the first group as working principles [43]. Working principles create the connections between the physical effect, material, and geometric structure. An example for this is the working principles of the electric motor that can be used as a solution of the function to convert electrical energy into mechanical energy.

This similarly applies for the area of information processing. In the area of software engineering, software patterns are utilized in order to save cooperating objects and classes in case these solve a general design problem. The patterns contain information on how they can be used and implemented in new situations. The solution description is made up of a structure and the partial behavior of each structural element. The domain of control engineering describes solution patterns on how a control loop is created, influenced or how the size of a path is measured or observed. Patterns of control engineering are primarily assigned to patterns of information processing. In doing so it must be observed that the patterns of control engineering can be concretized as patterns of software technique or as working principles. Besides the mentioned specifications of solution patterns, in the context of self-optimizing systems, working patterns for self-optimization come into play [50]. They are used for the implementation of self-optimization processes. Working patterns for self-optimization fulfill functions for self-optimization like autonomous planning, cooperating, acting and learning. The spectrum of the working patterns for self-optimization envelops the complete self-optimization process (1. Analysis of the actual situation, 2. Designation of system objectives and 3. Adaption of the System Behavior) [40].

As already mentioned the early design phases of self-optimizing systems require an effective cooperation and communication between all developers involved. To come up with such requirements we developed a uniform specification of solution patterns that is similar for all the disciplines involved. We structured the specification in six aspects with respect to the categories according to Alexander (1977), which is presented in Fig. 4.20 and will be explained in the following:

**Characteristics:** This Aspect describes the characteristics of the pattern. Because of the significant differences between the characteristics of the basic system and the information processing we will distinguish two subcategories of this Aspect. Examples for relevant characteristics of the basic system are usage conditions, geometry or material. In this context it is useful to note the environmental conditions, where the physical system elements can be used. The geometrical aspect implies the description of the approximate dimensions of the elements. Material characteristics in particular have to be specified for a compatible combination of solution patterns for the basic system. This aspect refers to similarities of the system elements and the working medium like fluid in hydraulic systems.

Similar to the patterns for the basic system, the information processing of the OCM is linked to some kind of usage conditions. More examples are processing speed or the type of calculation. So by generating a new SP for the information processing, the developer has to differentiate between hard real-time and soft real-time. The kind of calculations is based on different information methods. Generally we distinguish between mathematic relations and software code that represent an application's flow of commands.

**Functions:** This Aspect contains all those functions that the SP can realize. Thus this aspect expresses the problem description. The hierarchical structure facilitates the developers to assign a suitable SP for the underlying problem.

**Active Structure:** The aspect active structure is the core of the solution-description. This Aspect specifies which system elements are necessary in order to implement the SP and how those system elements are interrelated. To support the developer to handle the complexity of a self-optimizing system we designed a general structure according to the OCM. This structure shows the basic elements of a self-optimizing system and has to be modified and concretized for each problem. This general structure also clarifies the interface between all involved disciplines on the different levels of the OCM.

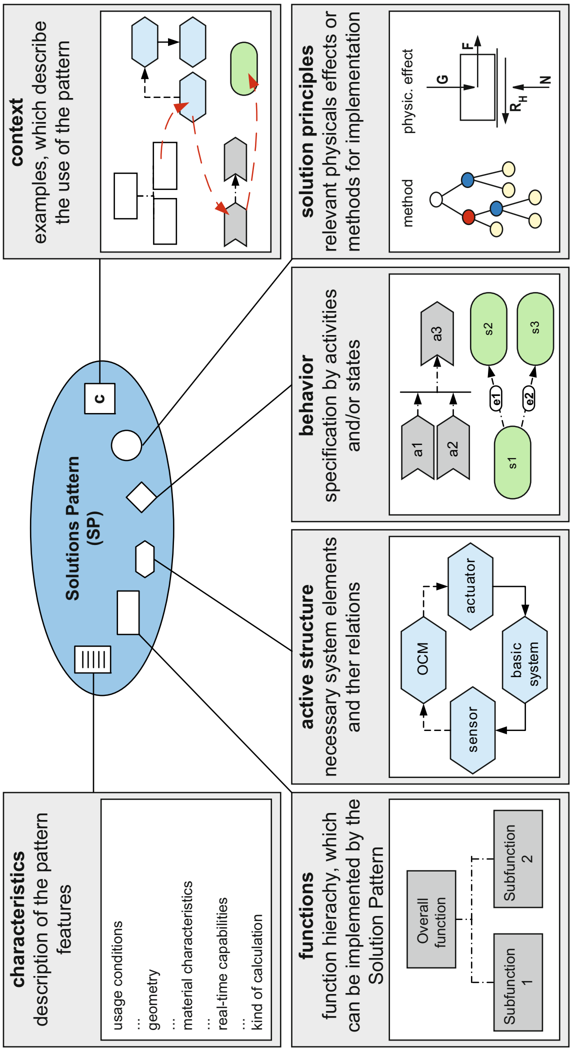


**Fig. 4.20** Uniform specification of solutipPatterns [10]

**Behavior:** This Aspect describes the behavior of the system elements. The aspect behavior is split into two sub-aspects. Behavior–activity describes the activities that are performed by the active elements during operations. So this sub-aspect has to be developed for patterns of the information processing. However, the behavior–state aspect models the possible sequences of states and states transitions of all system elements of the active structure. Thus this aspect has to be modified for each pattern to design self-optimizing systems.

**Implementation:** In the course of engineering, the implementation of solutions is generally based on certain method. However, an explicit definition that focuses the implementation of physical problems as well as information processing does not exist. According to the definition by Sauer (2006) [48], methods describe sequences of physical, chemical, biological or information processing application flows that are necessary to realize the defining functionality. So a method concretizes the abstractly formulated process, which describes the transformation of the operand from the initial state to the final state.

Therefore patterns of the basic system are generally based on physical effects. The Controller algorithms are based on mathematic relations like elementary transfer elements (e.g. P-element). However, the SPs for the RO are based on software algorithms. These are implemented in the CO model-based and behavior-based methods.

**Context:** A solution pattern is generally attached to a specific context, so this Aspect completes the uniform specification. Examples, which clarify the successful use of the pattern, have to be declared. The core of each pattern is the description of the underlying problem and the related solution. Therefore the aspects functions, active structure and behavior have to be modified for each example.

The presented specification is similar for all different problems within the domain-spanning conceptual design. Because of the increasing complexity of self-optimizing systems the following question is asked: Is it possible and advantageous to categorize solution patterns?

In order to design self-optimizing systems, we propose a categorization according to the generic composition of mechatronic systems that adjusts to the subdivision of information processing into several hierarchical levels, see Fig. 4.21. This division is well-grounded in cognitive science and enables the illustration of information processes which implement intelligence (cf. Chap. 1).

**Tool Support:** The structure of the above presented solution patterns allow the externalization and documentation of reusable solution knowledge. However, an efficient use of these patterns requires an appropriate computer support. Developers need a way to create new solution patterns, to store them in a repository, as well as an opportunity to integrate existing solution patterns in their current development process. A need for some kind of database, in which solution patterns and thus the knowledge of the experts can be stored, is apparent. Therefore, we developed a knowledge base for the systematic management of solution patterns, called "Solution Pattern Knowledge Base". The basic functionality is shown Fig. 4.22.

The solution pattern knowledge base is a central repository for all developers. Apparently, it is necessary to extend the functional range of a standard database, which only stores the information. In a standard database all users need detailed knowledge about the structure of the information and possible search methods. Instead, the Knowledge Base has to support in such a manner, that developers from different domains are able to recognize an appropriate solution pattern.

An information system that implements the pattern repository must support the collaboration of the solution patterns developers, which are e.g. experts within the field of artificial intelligence or mathematical optimization, and the engineers. Our aim is the storage of all required information in one single repository to enable the developers an access to domain-spanning solutions during the system design. The Knowledge Base is composed by several parts: database, which stores solution

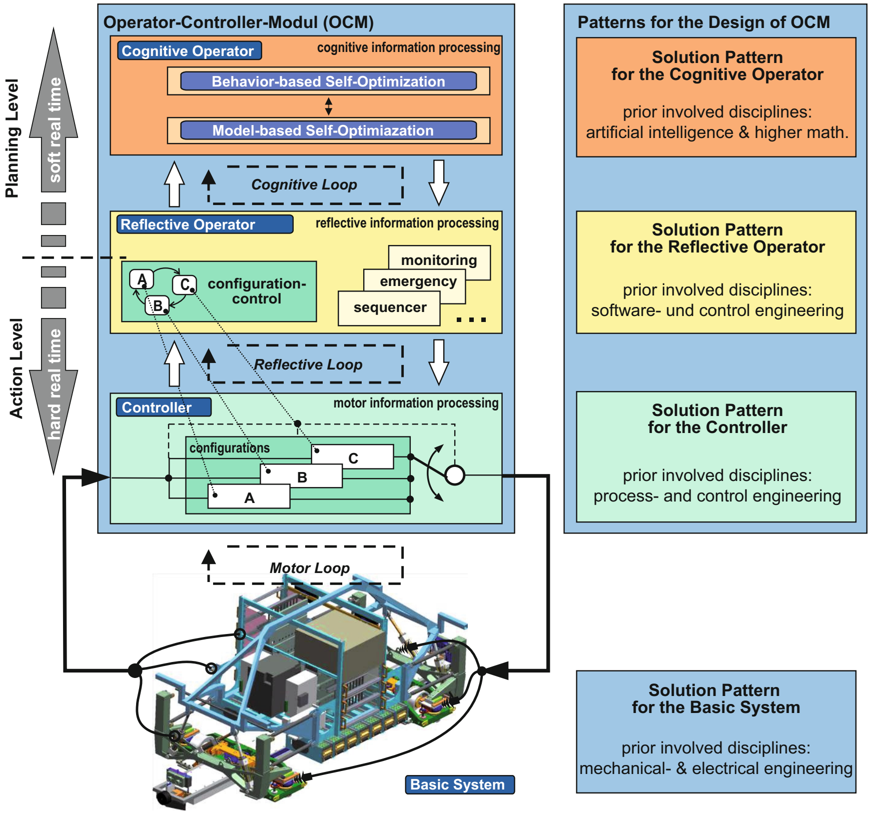

Operator-Controller-Modul (OCM)
Cognitive Operator
cognitive information processing
Behavior-based Self-Optimization
Model-based Self-Optimiazation
Cognitive Loop
Reflective Operator
reflective information processing
configuration-control
A
B
C
monitoring
emergency
sequencer
...
Reflective Loop
Controller
motor information processing
configurations
C
B
A
Motor Loop
Basic System
Planning Level
soft real time
Action Level
hard real time
Patterns for the Design of OCM
Solution Pattern for the Cognitive Operator
prior involved disciplines: artificial intelligence & higher math.
Solution Pattern for the Reflective Operator
prior involved disciplines: software- und control engineering
Solution Pattern for the Controller
prior involved disciplines: process- and control engineering
Solution Pattern for the Basic System
prior involved disciplines: mechanical- & electrical engineering

**Fig. 4.21** Classification of solution patterns according to the Operator-Controller-Module [10]

patterns and a catalog of functions; ontologies for the consideration of the semantic; inference-rules to combine solution patterns.

### *4.5.4 Systems Specification*

In order to support the engineers during the conceptual design of self-optimizing systems, especially the design of the information processing, we developed a design template for the active structure. Therefore it was necessary to identify the interrelationship between the cognitive functions and elements of the technical systems. In this context, the scientific cognitive view is transferred to the technically oriented OCM-architecture in the following. Figure 4.23 presents the transformation of the point of view from cognitive science to the technical OCM-architecture. It shows a general active structure that can be concretized by different flows. This general

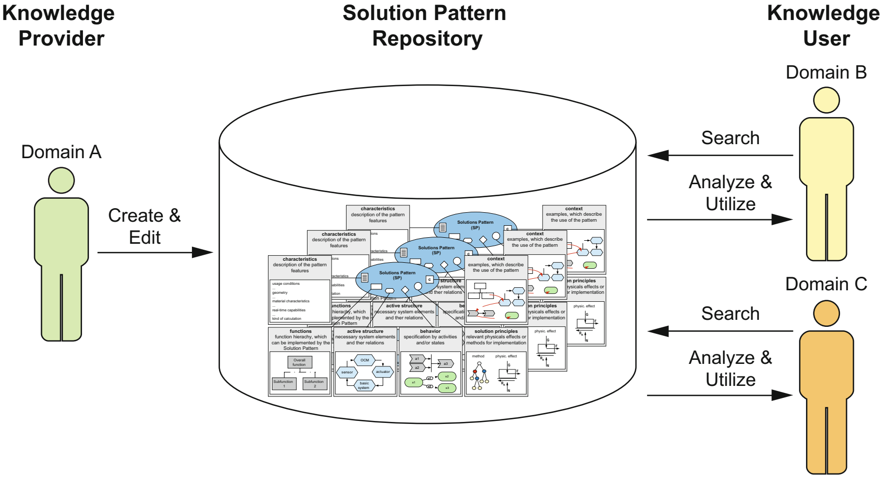


**Fig. 4.22** Basic concept of a knowledge base for the domain-spanning reuse of solution patterns (in accordance to [10])

structure also clarifies the interface between all involved disciplines of the different levels of the OCM.

**Basic System:** The physical system elements are located in the basic system. This can be subdivided into the following system elements: passive basic structure, hardware for data processing and energy supply. All energy flows of the basic system represent the attachment of single elements at the passive basic structure. The actuating and sensor elements are the interface for data processing.

**Controller:** The Controller (CO) realizes the non-cognitive control. The most important task of the Controller is to improve the system performance. This is implemented by adjusting disturbance values, which influence the basic system. From a process control point of view, the actuator elements are an integrative part of the setting device. It fulfills functions like e.g. "to balance system deviation". The Controller realizes a rigid interface between the actuating and sensor elements, whose action flow is known as a motoric loop

**Reflective Operator:** In the Reflective Operator (RO) the associative regulation, such as classical or operant conditioning, can be implemented by its corresponding algorithms. The RO can thus be used for learning control superimposed by the CO. The processing speed through the RO is subdivided in soft and hard real-time. The studies of existing systems have shown that associative functions run in soft real-time. Basically, the majority of the calculations in the RO run under hard real-time conditions. Its primary task is to generate reference values based on sensory input for regulation in the Controller. The data processing of the OCM has to split into several system elements that fulfill different functions (e.g. configuration management, monitoring or corrective element). Furthermore, a communication module is necessary to share important data or information with external systems.

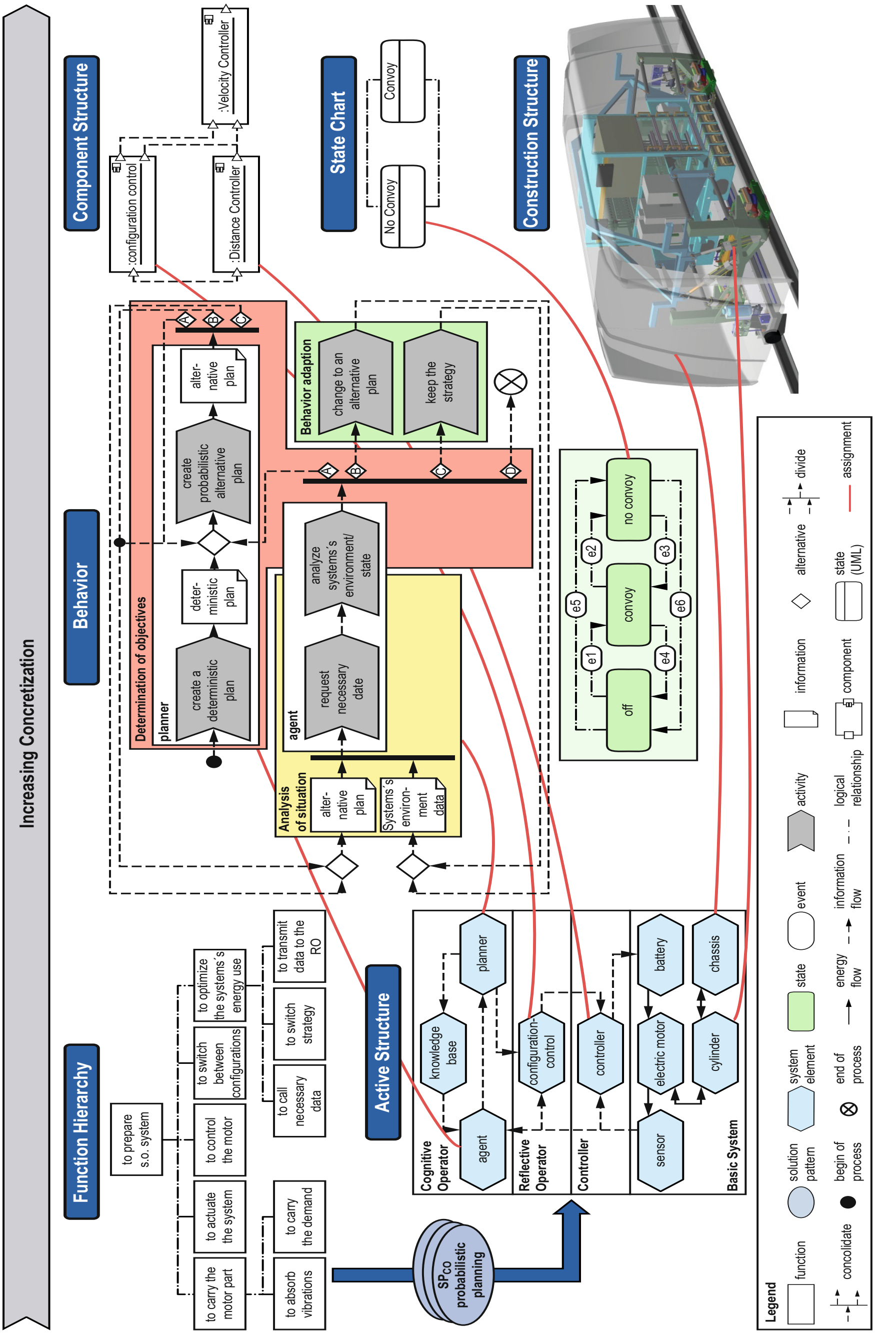


Fig. 4.23 Generic structure of intelligent technical systems based on the OCM-structure [10]

**Cognitive Operator:** The cognitive regulation is realized in the Cognitive Operator (CO). The calculation of the mathematical problems can run in offline or online mode. The system elements that are involved in the implementation of calculation methods are situation analysis, system of objectives and adapting system's behavior. The needed information to optimize are identified, received, tested and classified by the situation analysis. The keyelement of an autonomous intelligent system is a knowledgebase that stores the relevant information and generates internal knowledge. The system of objectives is the internal target system that plans the expected system behavior in certain situations. The results are transmitted to the element "adapting system's behavior". This system element does not directly access the regulation or the basic system with the appropriate actuators. Rather it adjusts the planned and optimized strategy in the RO. This coupling between RO and CO is called cognitive loop.

### Conceptual Design of Self-optimizing Systems with Solution Patterns Exemplified by Probabilistic Planning

According to Fig. 4.24 the use of patterns to develop self-optimizing systems starts on the left with the successful implementation of eligible solution patterns for specific sub-functions. In the next step the developer has to feed the aspects of the active structure and behavior to specify the complete system. Therefore it is generally necessary to modify these aspects of the basic pattern to achieve the different fundamental problem. The developed principle solution is the second intermediate result. At the beginning of concretization the aspects have to be transformed in domain-specific terminologies. So the process ends in an early specification of the information processing as a component structure and a state chart. Simultaneously the construction structure, based on the principle solution of the basic system, is developed.

In the following we will explain the solution pattern "**Probabilistic Planning**". The topic of the patterns, which is shown in Fig. 4.25, is the forward planning of possible situations when considering insecurity. The core of the planning is a decision tree consisting of an ideal path and several possible intersections, which lead to the same result. A condition is defined as the amount of the critical value of one or more state variables (conditional planning). The probabilistic planning contains the aforementioned limited planning and the execution monitoring. Furthermore, a new planning can be considered. The basic idea is to use the conditional planning as a standard practice. The new planning however represents a backup level for unpredictable and implausible events. In this case, more intersections have to be added to the decision tree. The requirements for the planning speed become more diverse depending on the situation, in which the new planning is necessary. Because of the fact, that the planning quality decreases if an alternative solution is found, the use of the backup level should be avoided as much as possible. In the following, the different aspects of the solution pattern "Probabilistic Planning" will be explained. The uniform specification characteristics are:

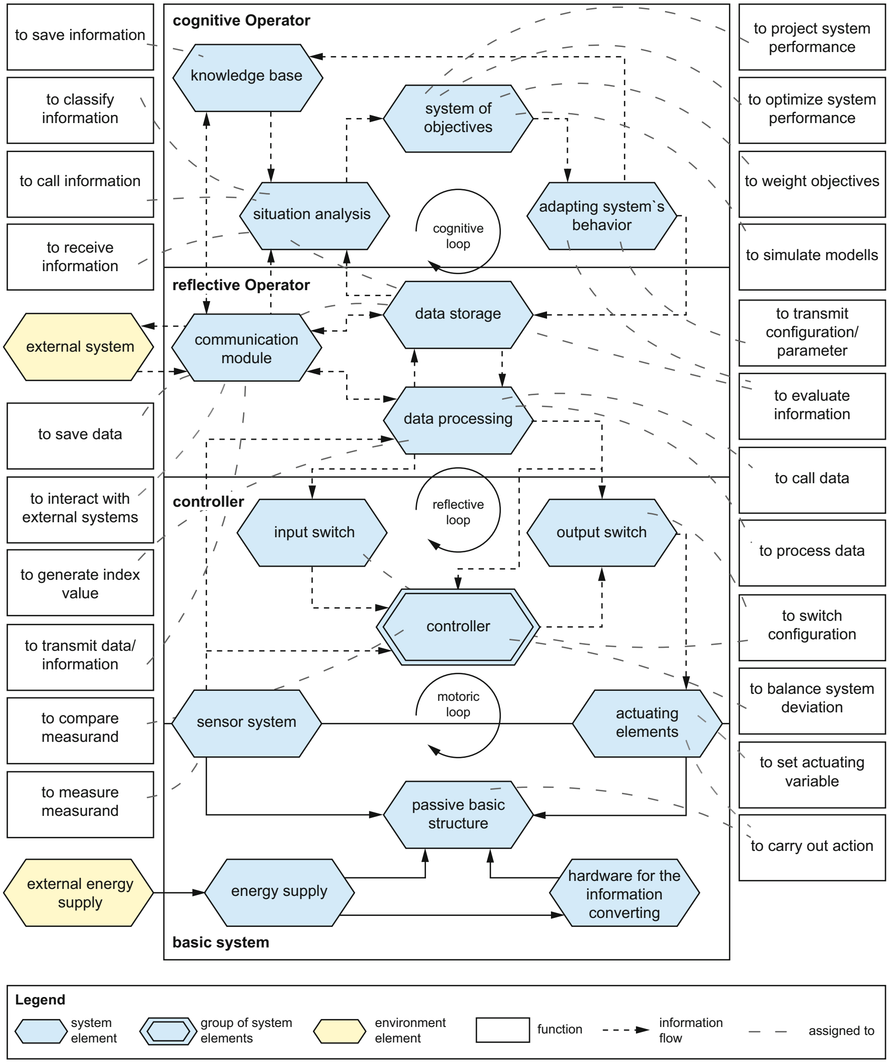


**Fig. 4.24** Design of self-optimizing systems with solution patterns [10]

*Working conditions:* The probabilistic planning is designed for mechatronic systems that don't always start from the same position and constantly change their positions, e.g. vehicle mobile robotics.

*Mode of calculation:* The processed values are discrete. This applies to both the conditional monitoring as well as for the execution monitoring.

*Real time capability:* The requirements for the planning time are diverse. The entry of an unexpected event has an effect on the quality of the new planning. The creation of the decision tree including the necessary intersections happens in soft real-time with a significant variation of the time limits.

*Modeling:* A holistic physical representation of the system‘s behavior is not necessary.

*Entity:* The system element‘s planner and agent work in a collective way.

*Modeling language:* The specification of this pattern is based on the detailed terminology PDDL (Planning Domain Definition Language).

**A functional description** of the SP probabilistic planning divides the overall function "evaluate probabilistic" into three sub-functions to analyze the situation, to determine objectives and to adapt behavior. The analysis of the situation calls all essential information of the knowledge base and preprocessed data of the Reflective Operator. The sub function to determine objectives is to predict using probabilistic planning and to compare situations in one step in the plan. The adaption of the behavior allocates a configuration in the next step in the plan. For each subsystem, one configuration exists. The configuration is then sent to the Reflective Operator.

**Active Structure:** The two central elements of the solution pattern "Probabilistic planning" are the agent and the planner. The tasks of the agent cannot be clearly assigned to a phase of the self-optimization process. The agent's participation is significant in the analysis of the situation based on the analysis of the environment as well as the analysis of the system state. The agent is active in the objective determination because the agent can evoke a new plan through his evaluation of a corresponding step in the plan. The planner on the contrary is assigned to determine objectives. The necessary information for the planner – which normally relates to models – resides in the knowledge base. With the help of this information, a planner can determine the probability distribution for all the discrete system states. The information is either generated in real-time or saved externally in the knowledge base. Additional data that is necessary is provided by the data call system element. It receives data from the data storage/memory of the RO and executes the system analysis along with the agent. The pattern group is complemented through the behavior adjustment.

**Behavior:** Directly after the beginning of the process, a deterministic plan is triggered. This occurs offline, because the system is incapable of action without an established strategy. The necessary decision tree contains a finite number of alternative branches that are created in a repeating loop. After the completion of the loop, it is verified if the plan is mature enough so that the system can begin the operation. At this point three deciding criteria are differentiated. If the conditions for the execution of an operation are not fulfilled, other branches are created. This way the normal as well as the new planning can be executed. However, if the condition is fulfilled, the system can begin the targeted aim. If the plan is not completed at this point the planning is resumed parallel to the executing system.

An essential task of the agent is to compare the next possible controlling point in the plan with the current environmental condition as well as with the system state. An evaluation is carried out after an analysis of the actual situation with the four deciding factors. If an unexpected event occurs that the existing branches do not consider the agent initializes a new plan through the planner. If however an alternative branch already exists for the event, the subsequent behavior adjustment will react correspondingly and an alternative will be set up accordingly to the configuration. If

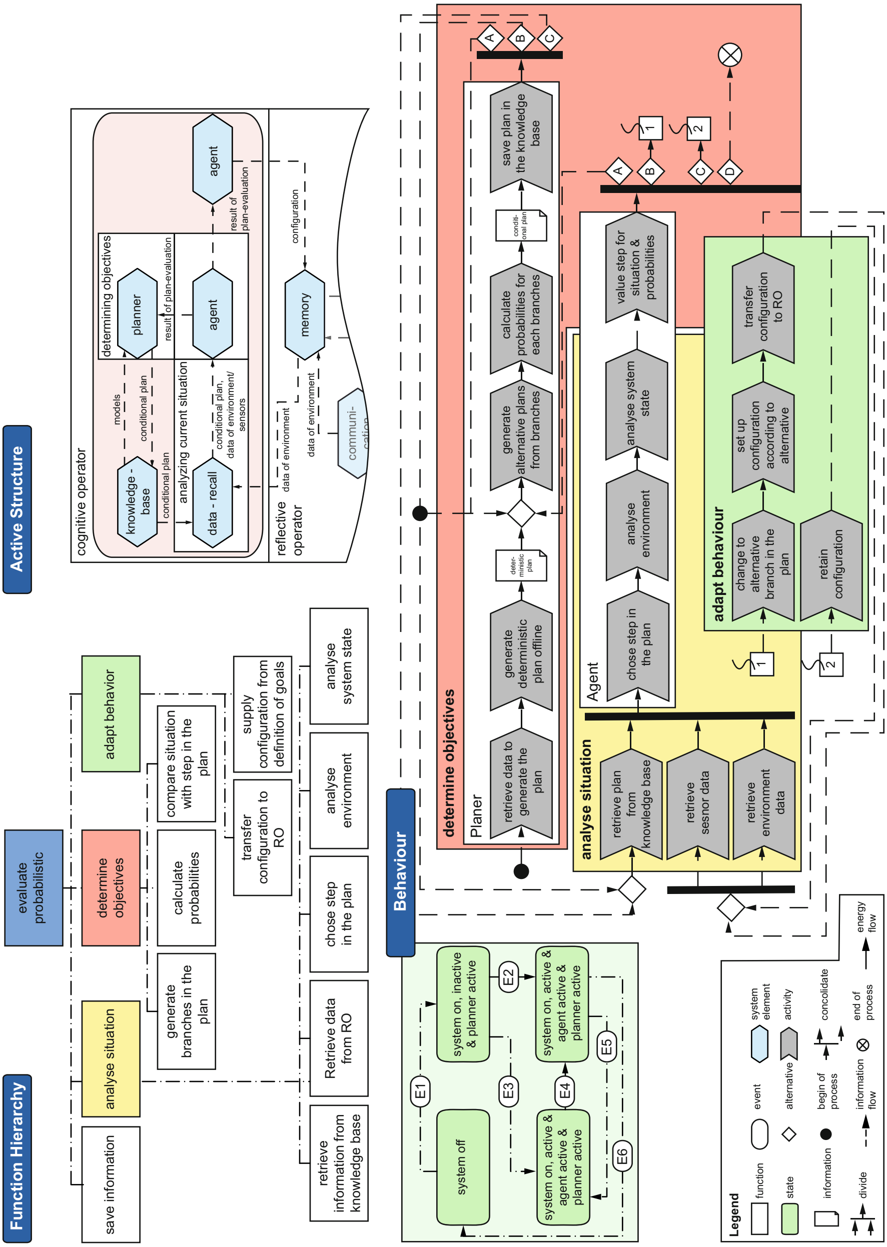

Function Hierarchy
evaluate probabilistic
save information
analyse situation
determine objectives
adapt behavior
generate branches in the plan
calculate probabilities
compare situation with step in the plan
transfer configuration to RO
supply configuration from definition of goals
retrieve information from knowledge base
Retrieve data from RO
chose step in the plan
analyse environment
analyse system state
Active Structure
cognitive operator
determining objectives
knowledge - base
planner
models
conditional plan
result of plan-evaluation
analyzing current situation
data - recall
agent
conditional plan, data of environment/ sensors
result of plan-evaluation
configuration
reflective operator
data of environment
memory
data of environment
communi-cation
Behaviour
system off
system on, inactive & planner active
system on, active & agent active & planner active
system on, active & agent active & planner active
E1
E2
E3
E4
E5
E6
determine objectives
Planer
retrieve data to generate the plan
generate deterministic plan offline
deter-ministic plan
generate alternative plans from branches
calculate probabilities for each branches
condi-tional plan
save plan in the knowledge base
A
B
C
analyse situation
retrieve plan from knowledge base
retrieve sesnor data
retrieve environment data
Agent
chose step in the plan
analyse environment
analyse system state
value step for situation & probabilities
A
B
C
D
1
2
adapt behaviour
change to alternative branch in the plan
retain configuration
set up configuration according to alternative
transfer configuration to RO
1
2
Legend
function
system element
event
state
activity
alternative
information
begin of process
concolidate
end of process
divide
information flow
energy flow

**Fig. 4.25** Main aspects of the solution pattern "Probabilistic Planning" [10]

the evaluation yields that the complete behavior proceeds according to plan, a check is performed to clarify if the targeted objective has already been reached or not. If the objective has been reached, the system is turned off. It is however necessary to finalize further steps in the plan to communicate to the behavior adaptation that the configuration should stay the same. The whole process runs recurrently until the objective of the plan is reached.

## 4.6 Product Structuring for Self-optimizing Systems

Rafal Dorociak and Jürgen Gausemeier

Another important method that supports the creation of the principle solution is the product structuring. Product structuring is an important mean to handle the complexity of a technical system [38]. The aim is to identify modules that form logical and functional units, which can be developed, tested, maintained and, if necessary, be exchanged autonomously by different teams. Thus, the product structure affects the whole product life-cycle.

Before we introduce the method by Steffen (2006) itself, two basic concepts have to be explained [53]. These are 1) the basic types of a development task with regard to product structuring and 2) design rules for product structuring.

**Basic types of development task:** In general, the product structure can be either modular, integral or a combination of both. Which product structure mechatronic systems and especially self-optimizing systems have, depends on a number of factors. These factors are, in particular, requirements on the product, on the product program and on the product development process. The analysis of several development tasks and their respective requirements has shown, that there is a number of criteria, according to which a development task can be described [27]. These criteria can be divided into three groups:

- **criteria with regard to the product:** These are: the size of the system, installation space, weight, performance data, recyclability, quality/reliability, availability, expandability and reconfigurability.
- **criteria with regard to the product program:** These are: number of market segments, planned product generations, quality of differentiation and variance of costs.
- **criteria with regard to the product development process:** These are: development effort, depth of the development and time of delivery.

Another result of the analysis of the development tasks is that, using the consistency analysis [21], they can be clustered in nine consistent combinations, which we call profiles [27]. As a consequence there are nine basic types of development tasks. As shown in Fig. 4.26 these are: 1) miniaturized product, 2) cost optimized mass product, 3) performance optimized single product, 4) complex miniaturized system, 5) system with numerous variants, 6) complex system with specialized modules, 7) mechatronic function module, 8) safety-intensive system and 9) reconfigurable system.

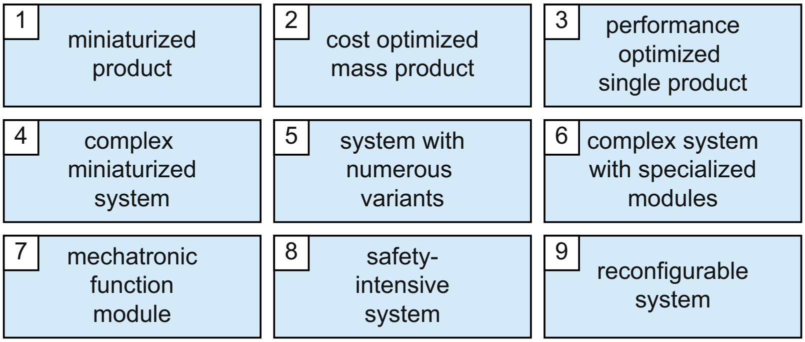
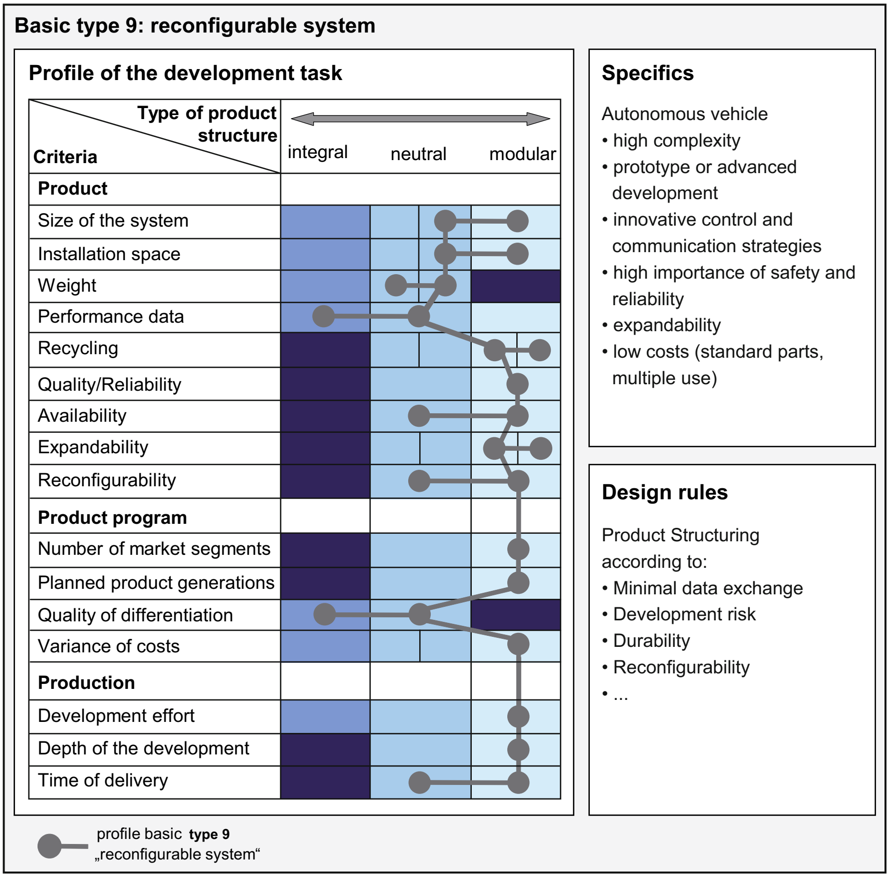

1
miniaturized product
2
cost optimized mass product
3
performance optimized single product
4
complex miniaturized system
5
system with numerous variants
6
complex system with specialized modules
7
mechatronic function module
8
safety-intensive system
9
reconfigurable system

**Fig. 4.26** Nine basic types of development tasks [27]

Figure 4.27 shows the profile of the basic type "9) reconfigurable system". An example of a system that complies to this basic type of development task is the RailCab. The profile is represented in a tabular form. The rows of the table are the aforementioned criteria. The cells describe the values of the respective criteria and whether they indicate a integral, modular or neutral type of the product structure. It is shown that reconfigurable systems are of relative high size and have high requirements when it comes to quality/reliability and availability as well as on expendability and reconfigurability. In addition, the breadth of the product program for such systems is high (i.e. a high number of markets has to be addressed and several product generations have to be planned). All in all, the values of the criteria indicate, that in most cases a modular product structure is recommended. Although, particular subsystems (modules) can certainly have an integral product structure, when some particular technical requirements have to be met.

**Design rules for the product structuring:** The basic types of a development tasks provide only a guideline. A direct adoption of the product structure for a whole class of systems is usually not possible (e.g. one product structure which can not be universally applied to all types of reconfigurable systems). Therefore, a number of design rules were defined, which support the developer by decision making in the conceptual design with regard to product structure relevant issues. In [27] 27 design rules were defined, which build eight categories. These are design rules for 1) product functionality, 2) disassembly/recyclability, 3) quality/reliability, 4) expandability, 5) standardization, 6) costs, 7) development and 8) manufacturing. The design rules address a number of properties of the product (disassembly/recyclability, expandability) and boundary conditions of the product development (standardization, manufacturing) into account. They are applied for the development of the partial models active structure, shape, etc. An example of a design rule is shown in Fig. 4.28. The main goal of this particular rule is to support reusability of the modules, which result from the product structuring.

For each basic type of development task a selection of such design rules has been defined [27]. Figure 4.26 shows a number of design rules, which are used for systems of the basic type "9) reconfigurable system". These are function fulfillment,



**Specifics**

Autonomous vehicle

- high complexity
- prototype or advanced development
- innovative control and communication strategies
- high importance of safety and reliability
- expandability
- low costs (standard parts, multiple use)

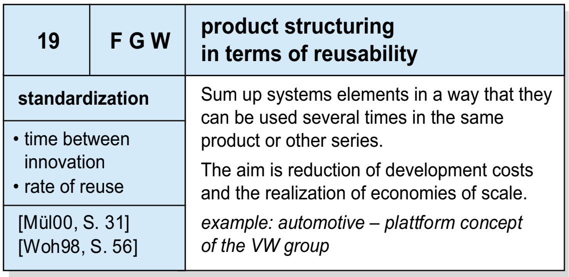
**Design rules**

Product Structuring according to:

- Minimal data exchange
- Development risk
- Durability
- Reconfigurability
- ...

**Fig. 4.27** Description of the characteristics of the basic development task "9) reconfigurable system" [27]

**Fig. 4.28** Example of a design rule: "product structuring in terms of reusability" [53]


19 F G W **product structuring in terms of reusability**

**standardization**

- time between innovation
- rate of reuse

[Mül00, S. 31]
[Woh98, S. 56]

Sum up systems elements in a way that they can be used several times in the same product or other series.

The aim is reduction of development costs and the realization of economies of scale.

*example: automotive – plattform concept of the VW group*

minimal data exchange, ability of testing and validation, durability, reconfigurability, user aspects, independence during further development, and development risk [27].

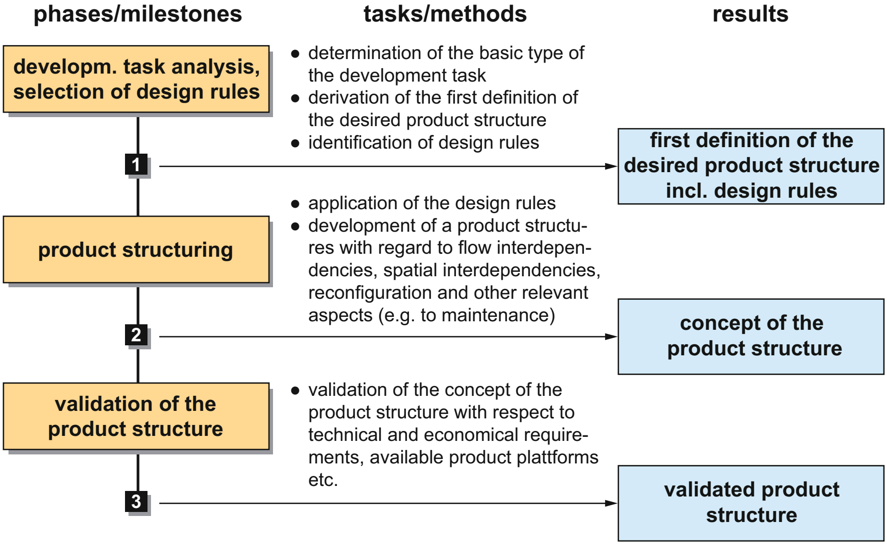

phases/milestones
tasks/methods
results
developm. task analysis, selection of design rules
• determination of the basic type of the development task
• derivation of the first definition of the desired product structure
• identification of design rules
1
first definition of the desired product structure incl. design rules
product structuring
• application of the design rules
• development of a product structures with regard to flow interdependencies, spatial interdependencies, reconfiguration and other relevant aspects (e.g. to maintenance)
2
concept of the product structure
validation of the product structure
• validation of the concept of the product structure with respect to technical and economical requirements, available product plattforms etc.
3
validated product structure

**Fig. 4.29** The procedure model of the method for the product structuring of self-optimizing systems based on the principle solution [53]

The method for the product structuring of self-optimizing systems consists of the following three essential phases [27] (Fig. 4.29):

**Phase 1 – development task analysis and selection of design rules:** First, the underlying development task is analyzed. The goal is to define the desired product structure for the system of interest. For this purpose, requirements on the product, the product program and the product development process are gathered according to the criteria described before. These are compared then with the profiles of the nine basic types of development tasks. Based on this comparison, one particular basic type of development task is chosen, which corresponds best to the development task. The result of this phase is a first definition of the desired product structure based on the profile of the chosen basic type. It serves as an orientation aid or "light house" during the further development (it can be compared to the "ideal concept" by Altschuller (2000) [28]). As explained in Sect. 4.6 there is a number of design rules assigned to each basic type of development task. For the system of interest the design rules are used, which correspond to the chosen basic type of development task.

The described approach has been validated on the RailCab. The analysis of the development task has shown that the development of the RailCab focuses on the validation of the applied technologies and the newly developed information technological processes (self-optimization). Design and efficiency of the prototype are less important. It is important that the drive and the active spring technology are accessible and modifiable during later test cases. For the validation of new processes in the field of information technology, additional properties are important. These are: autonomy of the included modules and system elements, learning ability, high control performance, and high safety requirements. In addition, the prototype has to be updatable. With respect to a later serial production, the mechanical components have to be reusable. Altogether the development task has the characteristics of basic development task "9) reconfigurable system" (Fig. 4.27). The corresponding design rules are: function fulfillment, minimal data exchange, ability of testing and validation, durability, reconfigurability, user aspects, independence during further development, and development risk.

**Phase 2 – product structuring:** The design rules selected in Phase 1 are applied in the course of the specification of the principle solution throughout the whole conceptual design, when design decisions with regard to product structuring are made. The design rules are used especially at the end of the conceptual design on the system level (before the beginning of the conceptual design on the subsystem level), as the subsystems (modules) of interest result from product structuring. In order to obtain modules, a number of established methods is used in combination. This is shown in Fig. 4.30. The starting point is the analysis of the specification of the principle solution (Fig. 4.30, (1)). Especially the partial models application scenarios, active structure and shape are concerned.

The information about system elements and their relationships (energy, material, information flows and spatial relationships) are extracted from the principle solution (mainly active structure and shape) and serve as input for the **Design Structure Matrix** (DSM) [14] (Fig. 4.30, (2)). With the resulting DSM the relationships between system elements are analyzed. The weighting of the different relationships is determined in accordance with the desired product structure and the chosen design rules. For product structuring, two particular views on the system are created. One focuses on its shape-oriented structure, the other one focuses on its function-oriented structure. Both views are then superimposed. Using DSM algorithms clusters of strongly dependent system elements are built. This happens semi-automatically. Some of the weights (values of the matrix) have to be modified manually in accordance with the underlying development task. The results are a product structure with regard to flow interdependencies and a product structure with regard to spatial interdependencies.

For self-optimizing systems, one aspect has to be particularly taken into account: self-optimizing systems have the ability to reconfigure. Hence, autonomous modules with disjoint functions and homogeneous interfaces have to be identified. For this purpose, the so called aggregation DSM and the **Reconfiguration Structure Matrix** (RSM) have been developed (Fig. 4.30, (3)). Both extend the DSM concept and use application scenarios of the system as input. For each application scenario a separate DSM is set up. Afterwards, the different DSMs are superimposed in two ways. Firstly, the aggregation of all DSMs is generated in a way, that all interrelationships are taken into account only once. The resulting aggregation DSM shows all possible connections within the system and allows the formulation of an adequate product structure. Secondly, the RSM is built. For this purpose, the frequency of the connections is taken into account by summing up the interrelations over all application scenarios. The resulting RSM allows the identification of system elements, which are only active in few application scenarios. This is a hint for reconfiguration potential. Such system elements could be integrated into autonomous additional modules.

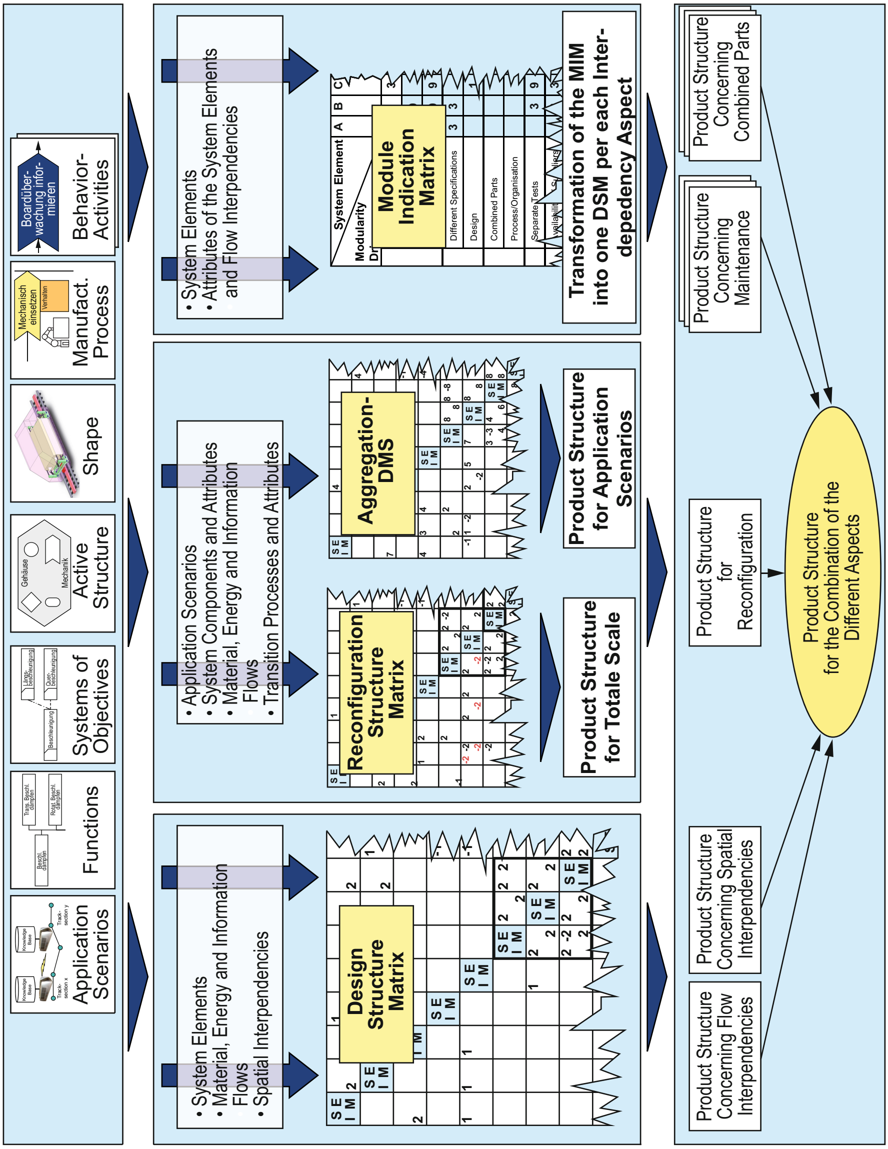

Application Scenarios
Functions
Systems of Objectives
Active Structure
Shape
Manufact. Process
Behavior-Activities
Boardüberwachung informieren
Mechanisch einsetzen
• System Elements
• Material, Energy and Information Flows
• Spatial Interpendencies
Design Structure Matrix
• Application Scenarios
• System Components and Attributes
• Material, Energy and Information Flows
• Transition Processes and Attributes
Reconfiguration Structure Matrix
Aggregation-DMS
Product Structure for Totale Scale
Product Structure for Application Scenarios
• System Elements
• Attributes of the System Elements and Flow Interpendencies
Module Indication Matrix
System Element
Modularity Driver
Different Specifications
Design
Combined Parts
Process/Organisation
Separate Tests
Transformation of the MIM into one DSM per each Interdepedency Aspect
Product Structure Concerning Flow Interpendencies
Product Structure Concerning Spatial Interpendencies
Product Structure for Reconfiguration
Product Structure Concerning Maintenance
Product Structure Concerning Combined Parts
Product Structure for the Combination of the Different Aspects

**Fig. 4.30** Interaction of the partial models with DSM, RSM, Aggregation-DSM and MIM [27]

System elements which are active in all application scenarios are integrated into basic modules of the system. The result of the application of the aggregation DSM and RSM is a product structure with regard to reconfiguration.

With the aggregation DSM and the RSM only flow and spatial interdependencies between system elements were taken into account. For the definition of the product structure, other aspects such as maintenance have to be considered, as well. A further refinement of the product structure is therefore necessary. We use the **Module Indication Matrix** (MIM) by Erixon (1998) [16] and its extension by Blackenfeld (1999) [39] for this purpose (Fig. 4.30, (4)). The MIM makes it possible to consider different properties of the system elements (e.g. maintenance intervals of a systems element) as well as partial model spanning cross-references (e.g. functions that a particular system elements concretizes). Which information is taken into account depends on the underlying development task and the selected design rules. The result of the analysis with a MIM with regard to the additional relevant aspects (e.g. with regard to maintenance).

Finally, the previously developed product structures are combined to the final product structure, which addresses all the aforementioned aspects (flow and spatial interdependences, reconfiguration, maintenance etc.) (Fig. 4.30, (5)) [27]. It forms the basis for planning of the activities in the further design and development phase. In particular, the resulting product structure integrates the two basic and mostly contradictory views of a shape- and function-oriented product structure. This is necessary, as both aspects are equally relevant for the development of mechatronic and especially self-optimizing systems.

During the conceptual design on the system level of the RailCab the design rules identified in Phase 1 were applied in an implicit way. The result is a first principle solution specified with the specification technique CONSENS presented in Sect. 4.1. At this stage the active structure for the RailCab consists of about 150 system elements. An explicit application of the design rules takes place at the beginning of the conceptual design on the subsystem level. For this purpose, flows (representing functional interdependencies) and spatial interdependencies are taken into account. Additionally the multiple usability of system elements is relevant. Two product structures are generated by using DSM. One with regard to flow interdependencies (function-oriented) and one with regard to spatial interdependencies (shape-oriented). Figure 4.31 shows these two structures (the function-oriented and the shape-oriented ones) and their relationship to the specification of the active structure for the RailCab. It is shown, that the two driving modules (front and rear) result from the shape-oriented product structure. They consist of one drive and one brake module and one axle that includes a Tracking Module as well as an Spring and Tilt Module. The Active Suspension Module, the Active Guidance Module and the Actuation Module are derived, from the function-oriented product structure.

As already explained, the initial product structure of the RailCab was refined by taking into account additional aspects. RSM and aggregation DSM are used to refine the Active Suspension Module. The MIM is used to analyze the aspects reusability and extensibility.

**Phase 3 – validation of the product structure:** Finally the developed product structure is validated. The core criteria of the validation are the level of compliance of the desired product structure to the underlying development task, technical and economical requirements as well as, if applicable, the available product platforms. If a need for improvement is identified, revisions of the product concept that support

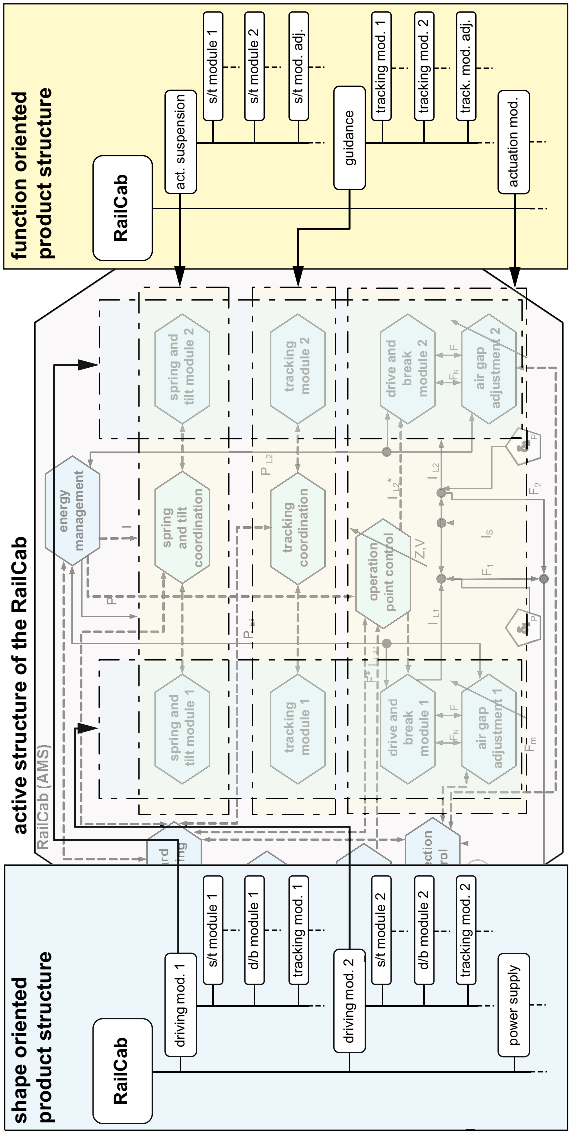


**Fig. 4.31** Comparison of initial shape- and function-oriented product structure [27]

a consequent realization of the desired product structure are initiated (e.g. modifications of the interfaces). Afterwards, the parallel domain-spanning design and development of the subsystems begins. The validated, development-oriented product structure of the RailCab is shown in Fig. 4.32 [27].

Product structuring is an important step in the development process for modern mechatronic and self-optimizing systems. It helps reduce the complexity and increase the quality of a system, but it also requires additional effort. A success factor is an adequate integration in the development process, by using established specification techniques, methods and tools. The presented approach shows, how this could be realized for mechanical engineering systems of tomorrow that may possess a high amount of information technology. The additional effort for product structuring during the conceptual design is profitable, compared to the costs of typically sub-optimal interfaces and high synchronization efforts which lead to time-intensive and costly iteration loops during the further design and development.

## 4.7 Early Probabilistic Reliability Analysis Based on the Principle Solution

Rafal Dorociak and Jürgen Gausemeier

Based on the domain-spanning description of the principle solution a number of analysis methods can be conducted. In this and the following section examples of such analysis methods will be shown. We begin with the method for the early probabilistic analysis of the reliability of a self-optimizing system based on its principle solution. It allows for first statements with regard to the reliability of the system in the early engineering phase of conceptual design. In particular, the weak points of the system with respect to reliability are found. For those weak points, detection measures and countermeasures are derived and implemented directly in the principle solution of the system. Altogether, the system under consideration is made more reliable in the early development stage.

The main input of our method is the domain-spanning specification of the principle solution (cf. Sect. 4.1). Following the recommendation of the CENELEC EN 50129 norm [13], our method uses two complementary reliability assurance methods FMEA (Failure Mode and Effects Analysis) [6, 25, 31] and FTA (Fault Tree Analysis) [6, 8, 32] interdependent. Some concepts known from the FHA (Functional Hazard Analysis) [58] method have been adapted, as well. This is especially relevant in, the use of a failure taxonomy for the identification of possible failures. By using these complementary methods, the completeness of the list of possible failure modes, failure causes and failure effects as well as of the specification of failure propagation is increased; both failure specifications are held mutually consistent.

Figure 4.33 shows the procedure model of our method; iterations are not shown.

**Phase 1 – specification of the principle solution:** The starting point are moderated workshops, where the experts from the involved domains work together in order to specify the system with the specification technique CONSENS as well as to analyze and optimize the principle solution with regard to reliability. In particular, the aspects functions, active structure, and behavior are described.

Our method has been performed for the RailCab. We will show some of the results for its Active Suspension Module. Each Active Suspension Module consists of three servo cylinders which dampen vibrations and tilt the vehicle body in curves. Figure 4.34 shows a cut-out of the partial model active structure for the servo cylinder of the Active Suspension Module. Each servo cylinder consists of a hydraulic cylinder, a 4/4-way valve, a servo cylinder regulation and a hydraulic valve regulation [46].

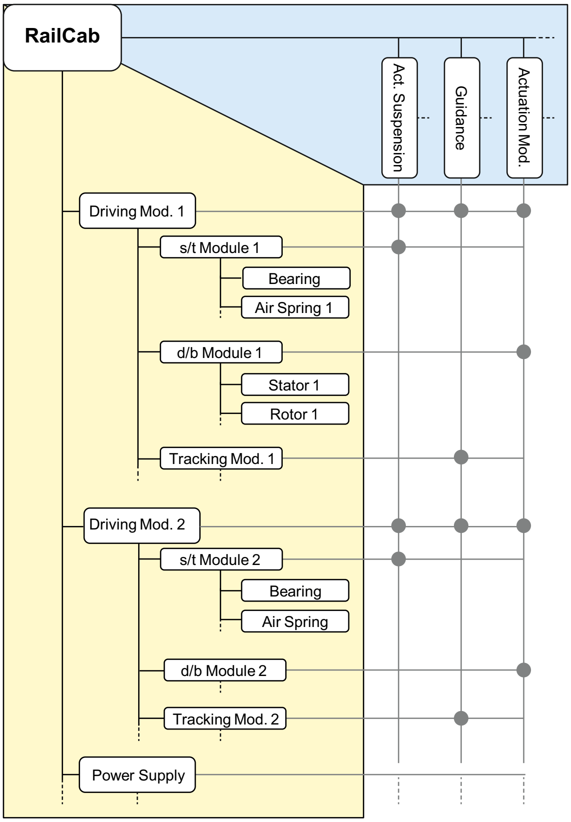


**Fig. 4.32** The product structure of the RailCab [27]

**Phase 2 – early FMEA based on the principle solution:** The system structure and the corresponding functions are automatically derived from the description of the partial models functions, active structure, and behavior which are recorded in the FMEA table. Failure modes, failure causes and failure effects are identified then. Checklists and failure taxonomies (e.g. one shown in Fig. 4.35) [17, 56] support the failure identification process. In addition, combinations of failure modes are identified, which can possibly occur together and have a negative impact on the system (pairs of failures, trios of failures, etc.). Failure modes and relevant failure mode combinations are recorded in the FMEA table. For each failure mode (and failure mode combination) the possible causes and effects are analyzed. Check lists can be used to accomplish this because they describe system elements known to be source of problems with regard to reliability [15]. A number of failure effects can be found by analyzing the principle solution for the system; this concerns the partial models active structure and behavior. A risk assessment of the failure modes, failure causes and failure effects takes place using the risk priority number (cf. the IEC 60812 norm [31]). Finally, detection measures and countermeasures are defined as

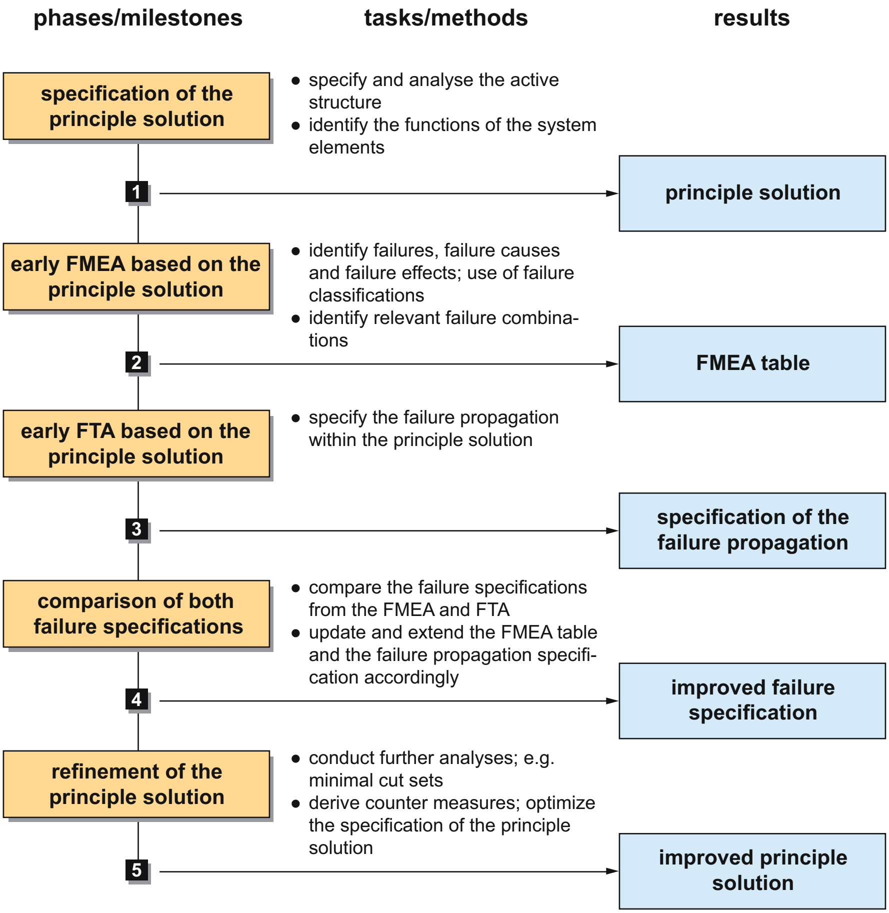

phases/milestones
tasks/methods
results
specification of the principle solution
• specify and analyse the active structure
• identify the functions of the system elements
1
principle solution
early FMEA based on the principle solution
• identify failures, failure causes and failure effects; use of failure classifications
• identify relevant failure combinations
2
FMEA table
early FTA based on the principle solution
• specify the failure propagation within the principle solution
3
specification of the failure propagation
comparison of both failure specifications
• compare the failure specifications from the FMEA and FTA
• update and extend the FMEA table and the failure propagation specification accordingly
4
improved failure specification
refinement of the principle solution
• conduct further analyses; e.g. minimal cut sets
• derive counter measures; optimize the specification of the principle solution
5
improved principle solution

**Fig. 4.33** The procedure model of the method for the early probabilistic analysis of the reliability of a self-optimizing mechatronic system

well as the corresponding responsibilities. This occurs similarly when compared to the classical FMEA. The FMEA table is updated accordingly.

The early FMEA method has been performed for the servo cylinder of the Active Suspension Module. A cut-out of the resulting FMEA table is shown in Fig. 4.36. Using the failure taxonomy by [17], the failure mode *hydraulic valve regulation provides no switch position for the 4/4-way valve* is found. This failure mode occurs, for instance, if the energy supply of the system element hydraulic valve regulation is interrupted. According to the FMEA the risk priority number for this case is 252. In order to eliminate or at least mitigate the failure mode, the energy supply of the hydraulic valve regulation should be monitored. One possible solution is to incorporate an additional monitoring system element into the principle solution. Then additional measures such as a redundant energy supply have to be implemented.

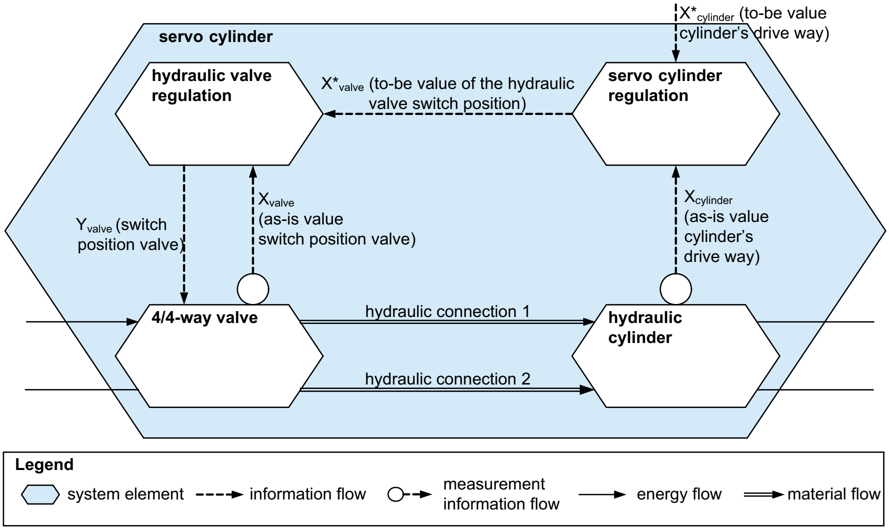

$X^*_{cylinder}$ (to-be value cylinder's drive way)
servo cylinder
hydraulic valve regulation
$X^*_{valve}$ (to-be value of the hydraulic valve switch position)
servo cylinder regulation
$Y_{valve}$ (switch position valve)
$X_{valve}$ (as-is value switch position valve)
$X_{cylinder}$ (as-is value cylinder's drive way)
4/4-way valve
hydraulic connection 1
hydraulic connection 2
hydraulic cylinder
Legend
system element
information flow
measurement information flow
energy flow
material flow

**Fig. 4.34** Active structure of the Active Suspension Module (cut-out)

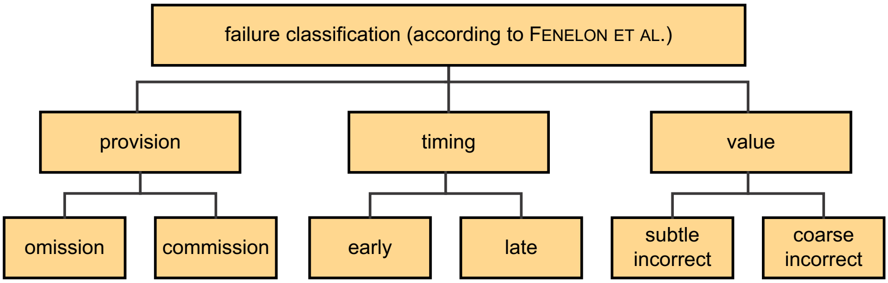


**Fig. 4.35** Failure classification (according to FENELON ET AL.(1994)) [17]

**Phase 3 – early FTA based on the principle solution:** The specification of the failure propagation within the principle solution is performed. The process is very similar to the traditional FTA. For each system element, its internal failures as well as incoming and outgoing failures are specified and related to each other.

In our application example, the specification of the principle solution is extended by the specification of failure propagation (cf. Fig. 4.37). For each system element the relationship between incoming, local and outgoing failures is described. For instance, the output $HV_2$ exhibits an undesired system behavior, if the internal failure "F1" or "F2" occur or one the input $HV_1$ is faulty. Based on such a description of the failure propagation a fault tree can be generated (semi)-automatically [8].

**Phase 4 – comparison of both failure specifications:** The FMEA table and the specification of the failure propagation both contain information about causal relationships between failures. Following the recommendation of the CENELEC EN 50129 [13], we use both methods in combination, to ensure a completeness of the

**Failure Mode and Effects Analysis (FMEA)**
**module: servo-cylinder**

| system element | function | failure mode | failure effect | s | failure cause | d | o | rpn | counter or detection measure | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| hydraulic valve regulation | regulate position of the valve | hydraulic valve regulation provides no switch position for the 4/4-way valve | valve does not change the pressure on the output anymore | 6 | servo cylinder regulation does not provide to-be valve slider position | 2 | 7 | 84 | monitor the outgoing communication towards the 4/4-way valve | ... |
| | | | | | hydraulic valve regulation is broken | 9 | 3 | 162 | generate a warning message, if needed | |
| | | | | | energy supply of the hydraulic valve regulation is interrupted | 7 | 6 | 252 | monitor energy supply | |
| 4/4-way valve | close the valve | valve closes in an unwanted manner | hydraulic valve stays in the current position | 8 | energy supply is interrupted | 7 | 4 | 224 | monitor energy supply | ... |
| | | | | | magnetic coil is damaged | 8 | 3 | 192 | generate a warning message, if needed | |
| servo-cylinder regulation | determine the as-is value for the cylinder lifting way | a wrong as-is value for the cylinder lifting way has been determined | control deviation e = $X^*_{cylinder}$ – $X_{cylinder}$ has a wrong value; the desired lifting way will not be accomplished | 9 | sensor is damaged | 8 | 3 | 216 | monitor as-is value of the cylinder lifting way $X_{cylinder}$ | ... |
| | | | | | cable break | 9 | 2 | 162 | | |
| | | | | | sensor has been wired in a wrong way | 3 | 2 | 54 | redundant design of the measuring system | |
| | | | | | wrong calibration | 2 | 5 | 90 | | |
| ... | ... | ... | ... | ... | | | | | | |

s: severity of the failure effect
d: detection probability of the failure cause
o: occurence probability of the failure cause
rpn: risk priority number (rpn = s*d*o)

**Fig. 4.36** FMEA table of the servo cylinder (cut-out)

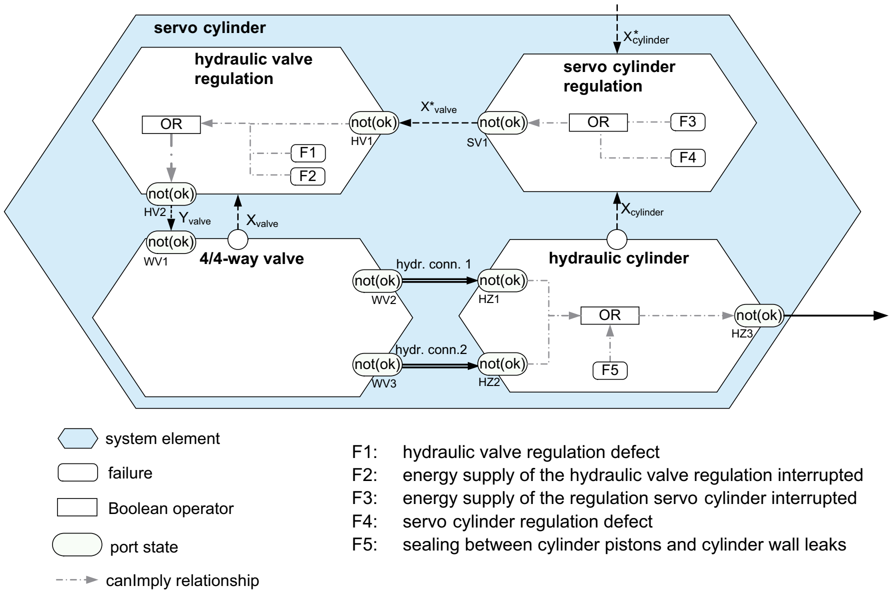
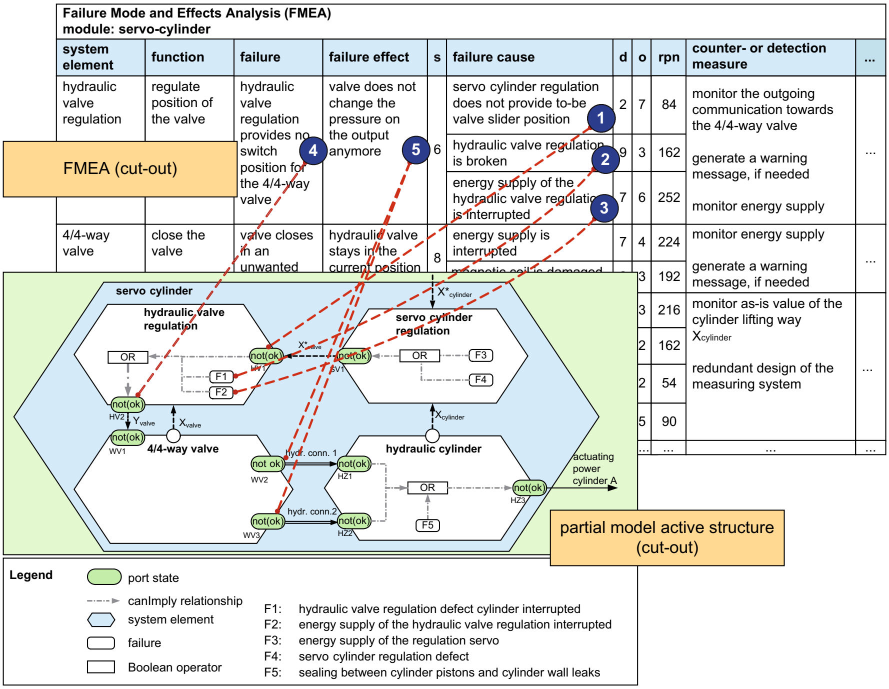

servo cylinder
hydraulic valve regulation
OR
not(ok)
HV1
F1
F2
not(ok)
HV2
Y_valve
X_valve
X*_valve
not(ok)
SV1
servo cylinder regulation
OR
F3
F4
X*_cylinder
X_cylinder
not(ok)
WV1
4/4-way valve
hydr. conn. 1
not(ok)
WV2
hydr. conn.2
not(ok)
WV3
hydraulic cylinder
not(ok)
HZ1
not(ok)
HZ2
OR
F5
not(ok)
HZ3
system element
failure
Boolean operator
port state
canImply relationship
F1: hydraulic valve regulation defect
F2: energy supply of the hydraulic valve regulation interrupted
F3: energy supply of the regulation servo cylinder interrupted
F4: servo cylinder regulation defect
F5: sealing between cylinder pistons and cylinder wall leaks

**Fig. 4.37** Specification of the failure propagation of the servo cylinder (cut-out)

failure specification. This can be achieved by comparing the information content of the FMEA and the failure propagation specification: e.g. failures and causal relationships between failures can potentially be found in the failure propagation specification, which have not been found during the FMEA and are thus not documented in the FMEA table; the FMEA table is updated accordingly. This also applies for the other comparison direction: For example, if a causal relationship between two failures (e.g. between a failure mode and a failure effect) has been recorded in the FMEA table, there has to be a corresponding causal relationship in the failure propagation specification. If this is not the case, the causal relationship is incorporated into the failure propagation specification. In the process, some additional failures which have not been specified can be found. The completeness of the identified failure modes, failure effects, failure causes as well as of the failure propagation specification is improved.

Figure 4.38 depicts the interrelation between both failure representations for the servo cylinder. The failure cause *servo cylinder regulation does not provide to-be valve switch position* from the FMEA table (cf. Fig. 4.38, (1)) corresponds to the port state not(ok) of the input HV1 of the system element hydraulic valve regulation. The failure causes *hydraulic valve regulation is broken* (2) and *energy supply of the hydraulic valve regulation is interrupted* (3) correspond to the internal failures F1 and F2 of the hydraulic valve regulation. The aforementioned failure causes (2) and (3) may lead to the failure *hydraulic valve regulation provides no switch position for the 4/4-way valve* (4); it is recorded in the FMEA table as well as in the specification



**Failure Mode and Effects Analysis (FMEA)**
**module: servo-cylinder**

| system element | function | failure | failure effect | s | failure cause | d | o | rpn | counter- or detection measure | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| hydraulic valve regulation | regulate position of the valve | hydraulic valve regulation provides no switch position for the 4/4-way valve | valve does not change the pressure on the output anymore | 6 | servo cylinder regulation does not provide to-be valve slider position | 2 | 7 | 84 | monitor the outgoing communication towards the 4/4-way valve | ... |
| | | | | | hydraulic valve regulation is broken | 9 | 3 | 162 | generate a warning message, if needed | |
| | | | | | energy supply of the hydraulic valve regulation is interrupted | 7 | 6 | 252 | monitor energy supply | |
| 4/4-way valve | close the valve | valve closes in an unwanted | hydraulic valve stays in the current position | 8 | energy supply is interrupted | 7 | 4 | 224 | monitor energy supply | ... |
| | | | | | magnetic coil is damaged | | 3 | 192 | generate a warning message, if needed | |
| | | | | | | | 3 | 216 | monitor as-is value of the cylinder lifting way $X_{cylinder}$ | ... |
| | | | | | | | 2 | 162 | | |
| | | | | | | | 2 | 54 | redundant design of the measuring system | |
| | | | | | | | 5 | 90 | | |
| | | | | | | ... | | ... | ... | ... |

Legend

- 🟩 port state
- canImply relationship
- system element
- failure
- Boolean operator

F1: hydraulic valve regulation defect cylinder interrupted
F2: energy supply of the hydraulic valve regulation interrupted
F3: energy supply of the regulation servo
F4: servo cylinder regulation defect
F5: sealing between cylinder pistons and cylinder wall leaks

**Fig. 4.38** Interrelation between the FMEA table and the specification of the failure propagation

of the failure propagation (port state not(ok) of the output HV2 of the hydraulic valve regulation).

According to the FMEA table there is a causal failure relationship between the failure *hydraulic valve regulation provides no switch position for the 4/4-way valve* (4) and the failure effect *valve does not change the pressure on the output anymore* (5). Although both failures were specified in the failure propagation model (input WV1 of the 4/4-way valve as well as inputs HZ1 and HZ2 of the hydraulic cylinder, respectively), the causal relationship between them has not been modeled. As a consequence, a thorough analysis was performed on the causal relationship. During this course the respective failure propagation path was modeled as well as an additional failure F6 (*valve position can no longer be changed mechanically; the valve slider stays in its current position*) (cf. Fig. 4.39).

**Phase 5 – refinement of the principle solution:** Both failure specifications are analyzed. For instance, the classical analyses known from the FTA field such as minimal cut sets are used [6]. In particular, the importance analysis is performed. For this purpose, the Bayesian network driven approach is used [9]; it enables the computation of the Fussell-Vesely importance measure. In this manner, the most critical system elements are identified. Detection measures and countermeasures are defined based on the analysis results. If possible, they are directly incorporated into

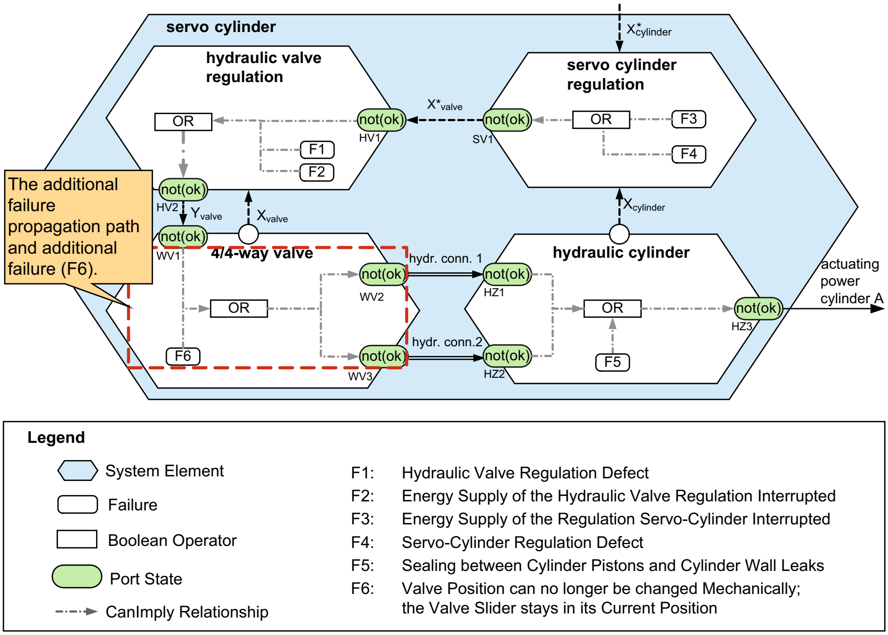

servo cylinder
hydraulic valve regulation
OR
not(ok)
HV1
F1
F2
$X^*_{cylinder}$
servo cylinder regulation
$X^*_{valve}$
not(ok)
SV1
OR
F3
F4
The additional failure propagation path and additional failure (F6).
not(ok)
HV2
$Y_{valve}$
$X_{valve}$
not(ok)
WV1
4/4-way valve
OR
F6
not(ok)
WV2
not(ok)
WV3
hydr. conn. 1
hydr. conn.2
$X_{cylinder}$
hydraulic cylinder
not(ok)
HZ1
not(ok)
HZ2
OR
F5
not(ok)
HZ3
actuating power cylinder A
Legend
System Element
Failure
Boolean Operator
Port State
CanImply Relationship
F1: Hydraulic Valve Regulation Defect
F2: Energy Supply of the Hydraulic Valve Regulation Interrupted
F3: Energy Supply of the Regulation Servo-Cylinder Interrupted
F4: Servo-Cylinder Regulation Defect
F5: Sealing between Cylinder Pistons and Cylinder Wall Leaks
F6: Valve Position can no longer be changed Mechanically; the Valve Slider stays in its Current Position

**Fig. 4.39** The extended failure propagation specification of the servo cylinder

the principle solution (e.g. redundancy, condition monitoring [37], etc.). Otherwise, they are recorded for further domain-specific design and development (e.g. test and simulation measures, etc.).

The result of the method is an updated principle solution for the system which is improved with regard to reliability. As a consequence, the reliability of the system under consideration is improved during the early development stage and a great number of time-intensive and costly iteration loops during the further development phases is avoided. The failure specifications and analysis results from the conceptual design are used in the further development phase of domain-specific design and development. The reliability analyses such as FTA and FMEA are performed again along with the concretization of the system.

In our application example, the specification of the failure propagation of the servo cylinder from the Active Suspension Module is translated into a Bayesian network. The translation algorithm proceeds as follows: for each system element its internal failures and port states of its inputs and outputs are translated into nodes of the Bayesian network. The relationships between them are represented as edges in the Bayesian network. The Conditional Probability Table (CPT) of the Bayesian network is then populated: for each value of variables associated to a node or a node state, its conditional probabilities are described, with respect to all combination of values associated to variables of the parent nodes in the network. To support the translation, a dictionary of translation rules has been developed [9], [26, D.o.S.O.M.S. Sect. 3.1] .

The result is a comprehensive Bayesian network, which describes the part of the system that is relevant for the examination of the chosen top event. Based on the Bayesian network some further analyses are performed [36]. In particular, the Fussell-Vesely importance measure is computed, i.e. it is determined with what probability a particular system element (failure cause) had led to a particular failure (the so-called posterior probability). The top event that we examine is *valve does not change the pressure on the output anymore* (corresponds to the port state WV2.not(ok)). Let us consider the state of the failure specification before the additional failure F6 and the corresponding propagation path were incorporated into the specification. The failure rates of the failures are shown in Tab. 4.4. Let us further assume, that the output WV2 of system element hydraulic valve regulation is in state not(ok), as this is our top event. According to the specification of the failure propagation (cf. Fig. 4.37) failures F1, F2, F3 and F4 contribute to this. Table 4.4 reports the Fussell-Vesely importance measure of each failure, i.e. the posterior probability of the contributing failures given the occurrence of the aforementioned failure. Failures F1 and F4 are especially important with importance greater than 28 %.

**Table 4.4** Failures, the failure rates and the Fussell-Vesely importance measure (before and after the failure specification had been extended) (Top-Event is WV2.not(ok))

| Failure | Failure rate (per hour) | Fussel-Vesely importance (before) | Fussel-Vesely importance (after) |
|---|---|---|---|
| F1 | $5.11 \times 10^{-7}$ | 0.2897 | 0.2416 |
| F2 | $4.02 \times 10^{-7}$ | 0.2279 | 0.1901 |
| F3 | $3.28 \times 10^{-7}$ | 0.1859 | 0.1551 |
| F4 | $5.23 \times 10^{-7}$ | 0.2965 | 0.2473 |
| F6 | $3.51 \times 10^{-7}$ | N/A | 0.1660 |

Now let us consider the extended specification of the failure propagation (including failure F6). The failure rate of failure F6 and the respective importance measures are shown in Tab. 4.4. The failures F1, F2, and F4 are of highest importance with importance of approximately 25 %.

All in all, by using our method, the completeness of the failure specification has been improved. Especially, failures and failure relationships were identified, which could have been easily omitted otherwise. In our application example, the failure F6 has been identified, the importance of which is quite high (approximately 17 %). Based on the failure specification, further analyses are conducted. Detection measures and countermeasures are then derived and, if possible, implemented directly in the principle solution. Altogether, the reliability of the system under consideration is improved during the early development stage.

## 4.8 Evaluation of the Economic Efficiency

Mareen Vaßholz

The decision to design a self-optimizing system is made in the early development phase, when the potential for the optimization of contradictory objectives is identified. In this case the developer has to determine whether these contradictions will be resolved in the further development by compromising or by using self-optimization. The technical feasibility and economical aspects also contribute to this decision, since the use of self-optimization can result in changing resource requirements for the development, production and operation when compared to a conventional mechatronic solution.

The **economic efficiency** of a system is given by the ratio of its evaluated monetary benefit to costs. The result is a dimensionless number. If it is exactly 1, neither profit nor loss is made [55]. In particular, the evaluation of the benefits of a self-optimizing system is challenging because it occurs during run-time of the system. The high complexity and the dynamics of the system in operation make the evaluation in the conceptual design phase difficulty. Existing methods to evaluate the costs and benefit do not meet this complexity [57]. The aim of the presented method is therefore to provide proof of the economic efficiency of self-optimizing product concepts during the early stage of the conceptual design based on the principle solution. This includes the evaluation of the two aspects costs and benefit for the company as well as for the customer. Furthermore it enables a comparison with conventional mechatronic solutions in order to select the most economical one. On this basis, the decision can be made whether the solution variant will be developed further in the design and development and eventually brought to market.

Figure 4.40 provides an overview of the phases and milestones of the method for the early estimation of the economic efficiency of a self-optimizing system based on the principle solution. The starting point for the development of a technical system are different product ideas, that can be beneficial for the customer and the company. Before the development of one of these product ideas is initiated, it needs to be clarified whether this can strategically benefit the company. In phase 1 the market for the product ideas is analyzed. For the stakeholder the qualitative benefit is identified by experts of the company. Based on these benefits the changes of the market performance of the company can be derived. For example through a competitive advantage, shares of sales from the competitors can be tapped and a higher revenue growth for the company results. In the case of promising expected revenue growth, the conceptual design of the product is initiated in phase 2. To be able to estimate the production costs in the subsequent phase, the production system is designed as well. Before the solution variant is developed further in the design and development phase, its economic efficiency needs to be proven. In order to pursue this we need to distinguish the economic efficiency for the company and for the customer. The quotient of the expected benefit (expected revenue growth and market price) and the anticipated costs (development, production and investment costs) describe the economic efficiency for the company. For the customer the solution variant is

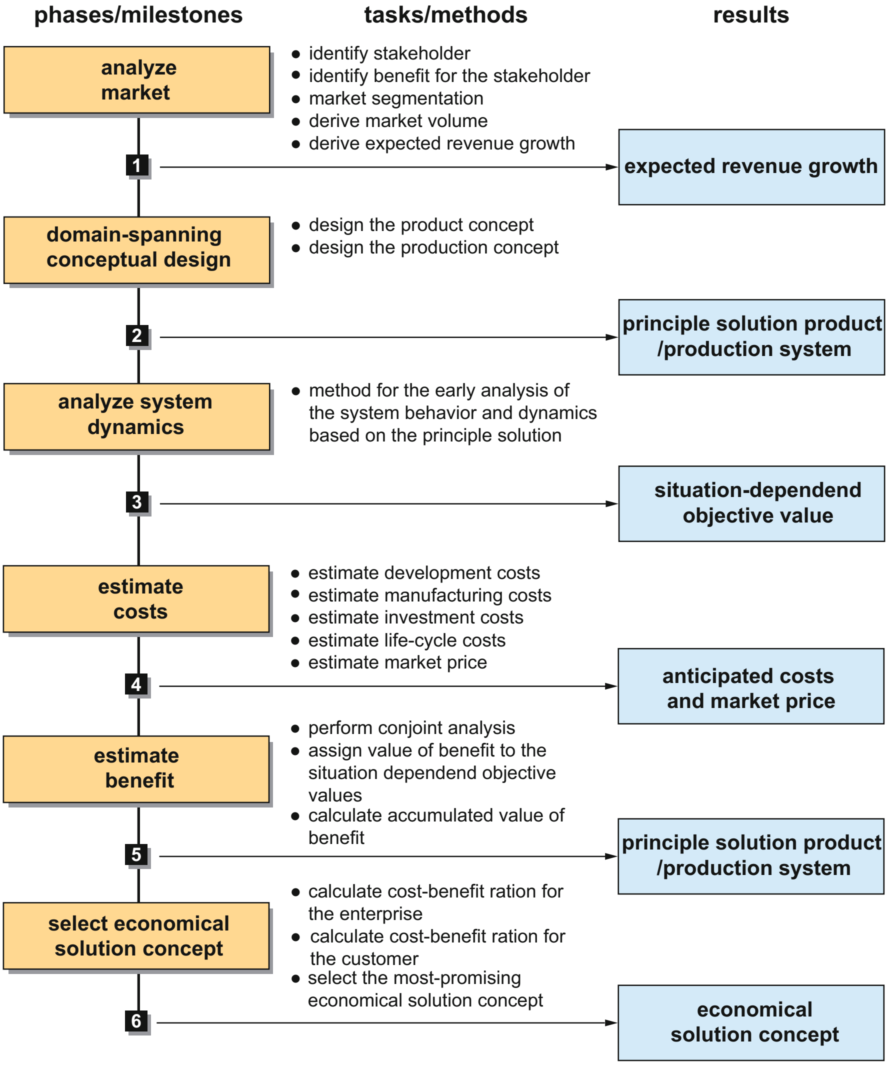

phases/milestones
tasks/methods
results
analyze market
• identify stakeholder
• identify benefit for the stakeholder
• market segmentation
• derive market volume
• derive expected revenue growth
1
expected revenue growth
domain-spanning conceptual design
• design the product concept
• design the production concept
2
principle solution product /production system
analyze system dynamics
• method for the early analysis of the system behavior and dynamics based on the principle solution
3
situation-dependend objective value
estimate costs
• estimate development costs
• estimate manufacturing costs
• estimate investment costs
• estimate life-cycle costs
• estimate market price
4
anticipated costs and market price
estimate benefit
• perform conjoint analysis
• assign value of benefit to the situation dependend objective values
• calculate accumulated value of benefit
5
principle solution product /production system
select economical solution concept
• calculate cost-benefit ration for the enterprise
• calculate cost-benefit ration for the customer
• select the most-promising economical solution concept
6
economical solution concept

**Fig. 4.40** Phase-milestone diagram for the early estimation of the economic efficiency of a self-optimizing system

economically efficient, if the quotient of the accumulated benefit in system operation to the life-cycle costs (purchase price, operation, maintenance and recycling costs) result in a value higher than one.

To be able to determine the benefit for the customer as well as the operation costs, the dynamics of the self-optimizing and the conventional mechatronic systems needs to be analyzed in phase 3. The result is the behavior of the systems in different operating situations. In the following phase the costs for the solution variants are estimated in phase 4. In phase 5 the benefit for the customer is estimated based on a conjoint analysis. The result is annotated to the simulation results of phase 3 and by accumulation, the benefit for the customer over the life-cycle of the system results. Finally the economic efficiency of each solution variant is calculated in phase 6 and the most promising one is selected and developed further.

**Phase 1 - analyze market:** The first phase is carried out before the conceptual design of the system ideas. Before time and money are invested for the development of a technical system, it needs to be clarified whether the development is advantageous for the company. Therefore the potential for the new system on the market needs to be analyzed and the expected revenue growth identified. This is the case, when stakeholders derive benefit from the system and thus are willing to buy it. Based on the approach by Freeman (1984) [19], the stakeholders are identified and analyzed. In this case a stakeholder is a group or an individual, who can affect or is affected by the achievement of the system to be developed. In the next step the identified stakeholder are categorized by the three attributes power, legitimacy and urgency due to Mitchell et al. (1997). They distinguish between eight categories: dormant, discretionary, demanding, dominant, dangerous, dependent, definitive stakeholder as well as nonstakeholder [41]. From this distinction preliminary indications can be conducted on how the stakeholder will benefit from the new system. For example dormant stakeholders could be customers of the competitors. In the case that the new system is brought to the market, they can be a potential customer for the new system, because their expectations are met. This leads to a growth in revenue for the company at the expense of the competitor. The identification of the benefit for the stakeholder by causal chains is performed in the next step. The qualitative benefit is collected in a stakeholder-benefit-matrix. The business is structured into market segments according to the segmentation criteria by Backhaus (2003) [2]. The identified stakeholder can be assigned to the market segments. The segments are compared to the market performance of the system ideas in a matrix. For each combination, the respective market volume, the company revenue, the sales growth in the previous year and the expected revenue growth are evaluated. The expected revenue growth can be predicted by the benefit that is resulting for the respective stakeholder. The result of the first phase is the expected revenue growth resulting from the development of the system for the company. On this basis, a decision is made whether the development of the system is advantageous or disadvantageous. In the first case the conceptual design of the idea is triggered. Otherwise the idea is rejected.

**Phase 2 - domain-spanning conceptual design:** A promising revenue growth is the trigger for the development of the system. In phase 2, the principle solution for the system is developed. To be able to estimate the production costs of the system, the respective production system needs to be designed in the early phase as well. For the development of the principle solution for the product the approach in Sect. 3.2 is used. In case a self-optimizing solution is not expressively required, a mechatronic and a self-optimizing solution is made. To develop the principle solution for the production system, part dimensions and material data are derived during the first step of determining the manufacturing requirements. Afterwards the active

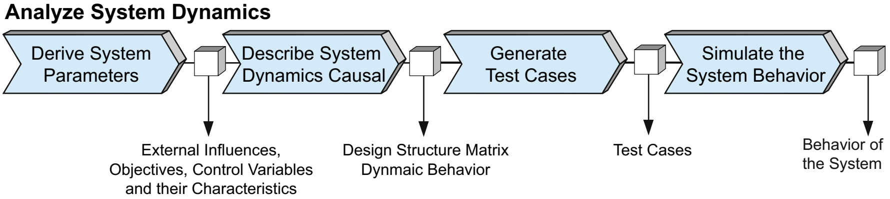

Analyze System Dynamics
Derive System Parameters
Describe System Dynamics Causal
Generate Test Cases
Simulate the System Behavior
External Influences, Objectives, Control Variables and their Characteristics
Design Structure Matrix Dynmaic Behavior
Test Cases
Behavior of the System

**Fig. 4.41** Four steps to analyze the system dynamics

structure and the building structure of the self-optimizing/mechatronic system are analyzed, to identify all system elements that need to be manufactured. Based on the structural connections, the process sequence for the manufacturing and the assembly process are set up. Based on the initial assembly sequence, the system elements are linked by assembly processes. Next, parts to be purchased and parts that need to be manufactured are determined. The parts to be manufactured are completed by the necessary manufacturing processes to produce them from raw materials or prefabricated parts. In this context adequate manufacturing technologies are chosen with respect to the deployed product technologies then the resources of the production system are determined. The resources realize the specified processes and are allocated to them, based on the chosen production technologies. The selection depends on the production requirements, this way, alternative resource combinations are obtained [22]. In the following phases the developed principle solution for the self-optimizing and mechatronic system will be analyzed regarding the economic efficiency, whereupon the concept of the production systems gives evidence for the production costs.

**Phase 3 - analyze system dynamics:** Because the benefits and costs of self-optimizing systems during operation results from the dynamics of the system, it needs to be analyzed. A particular challenge for self-optimizing systems is, that their behavior can not be predicted easily due to its interdependencies and the resulting high complexity. To reduce the complexity, we use a matrix based approach and map the dependencies of the control loop between external influences, control variables of the system and the objectives of the system in a Multiple Domain Matrix. By matrix multiplication the continuous **self-optimization process** can be simulated and the respective system state detected. This results in having the objective characteristics of the system state for each operating condition over time. For this purpose four steps need to be performed (cf. Fig. 4.41):

**Step 1 - derive system parameters:** To be able to simulate the system dynamics the principle solution (cf. Sect. 4.1) for the self-optimizing system and of the conventional mechatronic system needs to be analyzed to derive the necessary parameter. These are the external influences, control variables and the objectives of the system. External influences result from the flows in the aspect environment. This can be for example the desired speed of the RailCab by the customer. The control variables can be identified based on the active structure of the system, e.g. the

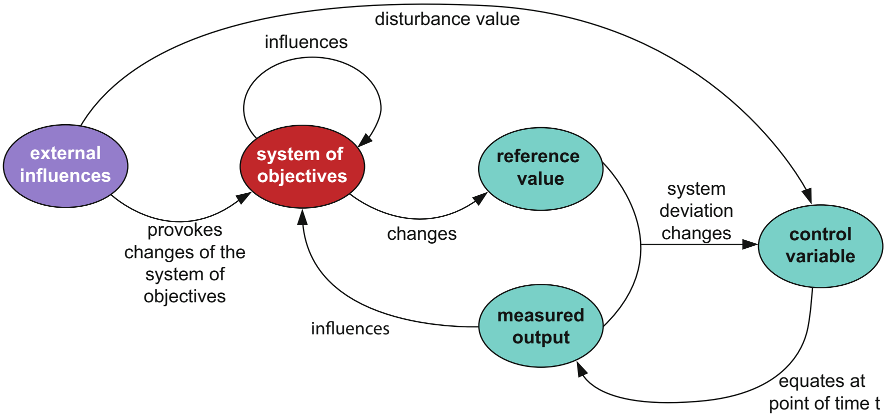


**Fig. 4.42** Closed-loop representing the dynamic behavior of a self-optimizing system

charge rate of the capacitor of the Hybrid Energy Storage System. An example for an objective of the RailCab is "minimize energy losses". For each parameter characteristics are determined. For instance the priority of the objective "minimize energy losses" can be qualitative low, high or very high.

**Step 2 - describe causal system dynamics:** The dependencies of the parameters in the closed-loop of the self-optimizing system derived in step 1 are presented in a causal diagram in Fig. 4.42. In the case that the external influences of the environment on the system changes a changed priority of the objectives of the system can be provoked. The objectives of the system of objectives are also influenced by itself and the current system state. The system state can be influenced by the environment by disturbing values. The self-optimizing system determines its objectives, if necessary. This leads to a changed reference value for the control strategy of the system. The deviation between reference value and measured system state can change the control variable and therefore the behavior of the system. This circumstance is described in the Multiple Domain Matrix in Fig. 4.43.

**Step 3 - generate test cases:** In this step the life-cycle of the system is represented by test cases consisting of different operating situations. For this, the situation of the aspect application scenarios is formalized. In the Design Structure Matrix (cf. Fig. 4.43, matrix No. 1) the consistency of the characteristics of the external influences were analyzed [21]. Different operating situations are derived from this matrix, consisting of a combination of characteristics of the external influences. These are clustered by similarity and assigned to the application scenarios, which describe one situation during the life-cycle of the system. Afterwards a sequence of application scenarios is generated and each one is provided with a time stamp. Thus test cases that represent the life-cycle result.

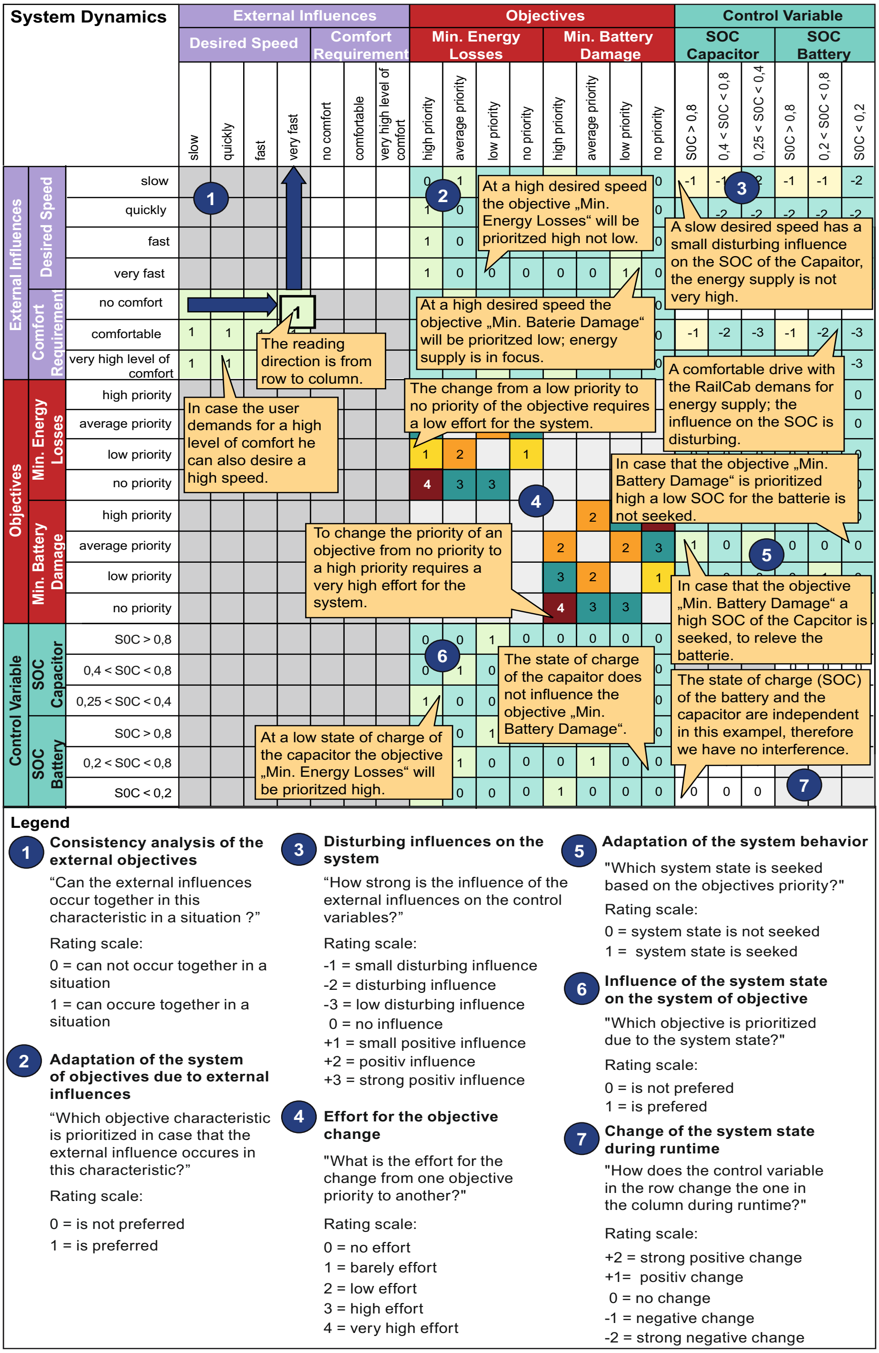
**Step 4 - simulate the system behavior:** For these test cases the dynamical behavior of the systems is simulated by matrix multiplication. For the self-optimizing system the self-optimization process is conducted as follows:

1. **Analyzing the current situation:** The self-optimization process is initiated, in case either the situation and therefore the external influences or the system state changes. The first case results by the sequence of the application scenarios in the test cases. The associated state vector for the external influences can be derived from matrix No. 1 (cf. Fig. 4.43). Furthermore it is examined whether this change leads to a changing system state because of disturbance variables (Fig. 4.43, matrix No. 3). The other case occurs for example when the capacity of the battery of the RailCab switches from one state to another, due to energy consumption of the system, as described in "4. Continuous system operation". The state vector for the control variable is derived.
2. **Determining the system's objectives:** The next step is to determine the objective of the system. The priority of the objectives is dependent on external influences (Fig. 4.43, matrix No. 2), the system state (Fig. 4.43, matrix No.6) and the current priority of the objectives (matrix No. 4). Each characteristic in the matrices demand a certain priority of the objectives. Out of these demands the Pareto optima for every objective prioritization is chosen and results in the state vector for the new objectives.
3. **Adapting the system behavior:** The alteration of the system of objectives for the system, demands a change of the systems behavior. Figure 4.43, matrix No. 5 presents the influence of the control variables by the priority of the objectives. The new state vector of the system can be taken and the system switches to another operation mode.
4. **Continuous system operation:** Since the system behavior can change during the operation, e.g. by energy consumption, continuous operation of the system is simulated as well. How the system state is changing over time is described in Fig. 4.43, matrix No. 8. The consumption, for example of the available energy is simulated over time. When the system state changes to another mode the self-optimization process is initiated again.

This procedure is conducted for all scenarios of the test cases. The respective priority of the objectives as well as the system state is recorded to be able to assign the operation costs and the situational benefit to each test case. For the conventional mechatronic system the simulation is conducted in a similar way, except that the changes of the objectives are limited to the defined control strategies.

**Phase 4 - estimate costs:** In this phase, the costs for the solution concepts are estimated. The costs for the company consist of the development, investment and production costs, which in turn are composed of various expenses. For the customer the life-cycle costs of the system are of interest. The presented method provides a guideline to estimate the relevant costs for the self-optimizing and mechatronic system.

To determine the **development costs**, the costs of all initiated processes for the development are required. For this purpose, the following questions must be answered: What do we do? How do we do it? and Who does it? [12]. To this end, the reference process (cf. Chap. 3) for the development of self-optimizing system is tailored due to the individual development task. This is accomplished using the



At a high desired speed the objective „Min. Energy Losses" will be prioritized high not low.

A slow desired speed has a small disturbing influence on the SOC of the Capaitor, the energy supply is not very high.

At a high desired speed the objective „Min. Baterie Damage" will be prioritized low; energy supply is in focus.

The reading direction is from row to column.

The change from a low priority to no priority of the objective requires a low effort for the system.

A comfortable drive with the RailCab demans for energy supply; the influence on the SOC is disturbing.

In case the user demands for a high level of comfort he can also desire a high speed.

In case that the objective „Min. Battery Damage" is prioritized high a low SOC for the batterie is not seeked.

To change the priority of an objective from no priority to a high priority requires a very high effort for the system.

In case that the objective „Min. Battery Damage" a high SOC of the Capcitor is seeked, to releve the batterie.

The state of charge of the capaitor does not influence the objective „Min. Battery Damage".

The state of charge (SOC) of the battery and the capacitor are independent in this exampel, therefore we have no interference.

At a low state of charge of the capacitor the objective „Min. Energy Losses" will be prioritized high.

**Legend**

1 **Consistency analysis of the external objectives**

"Can the external influences occur together in this characteristic in a situation ?"

Rating scale:

0 = can not occur together in a situation

1 = can occure together in a situation

2 **Adaptation of the system of objectives due to external influences**

"Which objective characteristic is prioritized in case that the external influence occures in this characteristic?"

Rating scale:

0 = is not preferred

1 = is preferred

3 **Disturbing influences on the system**

"How strong is the influence of the external influences on the control variables?"

Rating scale:

-1 = small disturbing influence

-2 = disturbing influence

-3 = low disturbing influence

0 = no influence

+1 = small positive influence

+2 = positiv influence

+3 = strong positiv influence

4 **Effort for the objective change**

"What is the effort for the change from one objective priority to another?"

Rating scale:

0 = no effort

1 = barely effort

2 = low effort

3 = high effort

4 = very high effort

5 **Adaptation of the system behavior**

"Which system state is seeked based on the objectives priority?"

Rating scale:

0 = system state is not seeked

1 = system state is seeked

6 **Influence of the system state on the system of objective**

"Which objective is prioritized due to the system state?"

Rating scale:

0 = is not prefered

1 = is prefered

7 **Change of the system state during runtime**

"How does the control variable in the row change the one in the column during runtime?"

Rating scale:

+2 = strong positive change

+1= positiv change

0 = no change

-1 = negative change

-2 = strong negative change

**Fig. 4.43** Causal description of the system dynamics with a Multiple Domain Matrix

framework for a self-optimizing process development of advanced mechatronic systems by Kahl (2013) [33] (cf. Sect. 3.4). For each process step it can be determined, which developer performs it, how long it will take and what potential additional expenses to the personnel costs will arise. The required personnel costs plus other expenses form the development costs.

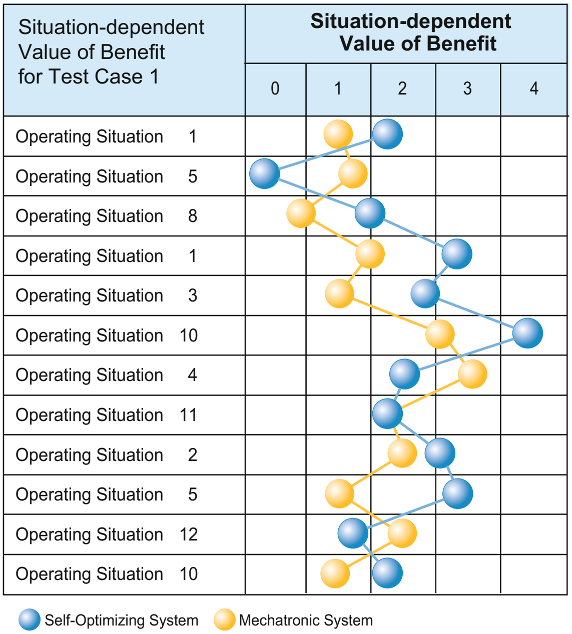
The **investment costs** for the company result from the procurement of material resources and from training costs for the developer in self-optimization specific expertise. For self-optimizing systems, this may result in part from the provision of new production plants, and also from the procurement of test bed and platforms. The investment costs for the production system can be derived from the principle solution for the production system. The necessary test beds and test platforms as well as training for the developers are associated with the solution pattern for self-optimization. Depending on the selected solution patterns, the necessary investments for the system test can be estimated.

The core of the product-related costs are the **production costs**. These consist of the basic mechanical structure by the material, manufacturing, assembly and testing costs. The manufacturing and assembly costs can, for example, be determined by the process costs based on the principle solution for the production system (cf. [42]). The production costs for the electronic components result in addition to the material and manufacturing costs from the so-called yield-loss costs as well as test costs [49]. To calculate the production costs for the software components, the staff required for the implementation, integration, and testing in personnel-months is included. In addition, there are charges which are incurred in the administration and sales which can not be directly associated with the system. These are shown in the surcharge calculation of overhead rates and added to the production costs. This results in the cost for the company for the product [12]. Based on these the market price can be calculated.

The **life-cycle costs** for the customer include the purchase price, the operating costs of the system as well as maintenance and recycling costs. The purchase price corresponds to the market price set by the company earlier in this phase. The operating costs of the system can be derived based on the simulated behavior of the system in phase 3. Over all operating situations, the changes of the system behavior were simulated. For each modification switching costs result. For the simulation these costs are taken as a factor. Furthermore, the consumption of the system has been simulated as well based on these factors. By the expertise of the developer these factors can be transform to monetary costs. The maintenance cycles can be estimated through the product life-cycle based on the simulation. Disposal costs are estimated based on the active structure and the expertise of the developer.

**Phase 5 - estimate benefit:** The benefit of a self-optimizing system during operation must be evaluated in terms of its situational dependency. The behavior of the system in different operating situations has already been simulated in phase 3. In this phase the benefit that results from certain behaviors of the system for the customer is identified with the traditional conjoint analysis [3]. In the first step the system properties and their characteristics to be queried are determined. For this purpose,

**Fig. 4.44** Roadmap for the value of benefit in different operating situations



the operating situations with the highest probability, the value of the system of objectives in the situation, as well as the operating costs for the situation are chosen. These are prioritized during a survey. Furthermore the price that the customer is willing to pay for the system should be queried in order to determine possible prices for the system based on the desired profit margins.

In the second step, the survey design is created. With the aid of the profile method, the stimuli which means the combinations of property characteristics are created. Then the number of stimuli is determined and a reduced design, which makes an evaluation manageable, is designed. The selected stimuli and operating situations are clearly described.

Then the stimuli are evaluated by the focus group. This group is selected by stakeholders in regard to the relevant market segments. The respondents are asked to prioritize the stimuli so that the resulting ranking order matches their personal preferences. Upon completion of the survey, partial values of benefit are determined for all property characteristics based on the empirically determined ranking-data. From this, the total values of benefit for all stimuli and the relative importance of each property can be derived. The aggregation of values of benefit is achieved using cluster analysis [21].

Finally, the values of benefit are assigned to the operating conditions for each test case and the situation-dependent benefit of each alternative solution is presented in a benefit-roadmap (cf. Fig. 4.44). The cumulative benefit of a the solution variant is derived from the weighted sum of the partial values of benefit. The monetary benefit is derived from the survey, based on the computed preferred price for the system and expenditure for a situation.

**Phase 6 - select economical solution concept:** The expected benefit and costs for the company and the customer are compared and the economic efficiency of each solution is calculated. The comparison provides the basis for decisions on the selection of the most economical solution variant that will be developed further in the domain-spanning design and development (cf. Sect. 3.3).

## References

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdhlking, I., Angel, S.: A Pattern Language. Oxford University Press, Oxford (1977)
2. Backhaus, K.: Industriegütermarketing, 13th edn. Vahlen, München (2003)
3. Backhaus, K., Erichson, B., Plinke, W., Weiber, R.: Multivariate Analysemethoden - Eine anwendungsorientierte Einführung, 13th edn. Springer, Heidelberg (2011)
4. Bertsche, B.: Reliability in Automotive and Mechanical Engineering - Determination of Component and System Reliability. Springer, Heidelberg (2008)
5. Birkhofer, H.: Analyse und Synthese der Funktionen technischer Produkte. Ph.D. thesis, Technische Universität Braunschweig, VDI-Verlag, Düsseldorf (1980)
6. Birolini, A.: Reliability Engineering - Theory and Practice, 5th edn. Springer, Heidelberg (2007)
7. Clark, N.: The Airbus Saga - Crossed Wires and a Multibillion-euro Delay - Business - International Herald Tribune, `http://www.nytimes.com/2006/12/11/business/worldbusiness/11iht-airbus.3860198.html?pagewanted=all` (accessed July 1, 2013)
8. Deyter, S., Gausemeier, J., Kaiser, L., Poeschl, M.: Modeling and Analyzing Fault-Tolerant Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
9. Dorociak, R.: Early Probabilistic Reliability Analysis of Mechatronic Systems. In: Proceedings of the Reliability and Maintainability Symposium (2012)
10. Dumitrescu, R.: Entwicklungssystematik zur Integration kognitiver Funktionen in fortgeschrittene mechatronische Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 286, Paderborn (2011)
11. Ehrlenspiel, K.: Integrierte Produktentwicklung, 2nd edn. Carl Hanser Verlag, München (2003)
12. Ehrlenspiel, K., Kiewert, A., Lindemann, U.: Cost-Efficient Design. Springer, Heidelberg (2007)
13. for Electrotechnical Standardization (CENELEC), E.C.: CENELEC EN 50129: 2003. Railway Applications - Communication, Signalling and Processing Systems - Safety Related Electronic Systems for Signalling. Norm (2003)
14. Eppinger, S., Whitney, D., Smith, R., Gebala, D.: A Model-Based Method for Organizing Tasks in Product Development - Reserarch in Engineering Design. Springer, Heidelberg (1994)
15. Ericson, C.: Hazard Analysis Techniques for System Safety. John Wiley & Sons, Hoboken (2005)
16. Erixon, G.: Modular Function Deployment - A Method for Product Modularization. Ph.D. thesis, Royal Institute of Technology, KTH, Stockholm (1998)
17. Fenelon, P., McDermid, J.A., Nicolson, M., Pumfrey, D.J.: Towards Integrated Safety Analysis and Design. ACM SIGAPP Applied Computing Review 2(1), 21–32 (1994)

18. Frank, U.: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 26, Paderborn (2006)
19. Freeman, R.E.: Strategic Management – A Stakeholder Approach. Pitman, Marschfield (1984)
20. Friedenthal, S., Steiner, R., Moore, A.C.: Practical Guide to SysML: The Systems Modeling Language. Elsevier, Amsterdam (2008)
21. Gausemeier, G., Plass, C., Wenzelmann, C.: Zukunftsorientierte Unternehmensgestaltung - Strategien, Geschäftsprozesse und IT-Systeme für die Produktion von morgen. Carl Hanser Verlag, München (2009)
22. Gausemeier, J., Brandis, R., Kaiser, L.: Integrative Conceptual Design of Products and Production Systems of Mechatronic Systems. In: Procedings of the Workshop on Research and Education in Mechatronics, Paris (2012)
23. Gausemeier, J., Dorociak, R., Pook, S., Nyssen, A., Terfloth, A.: Computer-Aided Cross-Domain Modeling of Mechatronic Systems. In: Proceedings of the International Design Conference, Dubrovnik (2010)
24. Gausemeier, J., Frank, U., Donoth, J., Kahl, S.: Specification Technique for the Description of Self-optimizing Mechatronic Systems. Research in Engineering Design 20(4), 201–223 (2009)
25. Gausemeier, J., Kaiser, L., Pook, S.: FMEA von komplexen mechatronischen Systemen auf Basis der Spezifikation der Prinziplösung. ZWF 11 (2009)
26. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
27. Gausemeier, J., Steffen, D., Donoth, J., Kahl, S.: Conceptual Design of Modularized Advanced Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
28. Gimpel, B., Herb, R., Herb, T.: Ideen finden, Produkte entwickeln mit TRIZ. Hanser Verlag, München (2000)
29. Greenyer, J.: Scenario-based design of mechatronic systems. Ph.D. thesis
30. Harel, D., Maoz, S.: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. Software and System Modeling 7(2), 237–252 (2007)
31. International Electrotechnical Commission (IEC): Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects Analysis (FMEA), IEC 60812 (2006)
32. International Electrotechnical Commission (IEC): Fault Tree Analysis (FTA), IEC 61025 (2006)
33. Kahl, S.M.: Rahmenwerk für einen selbstoptimierenden entwicklungsprozess fortschrittlicher mechatronischer systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagschriftenreihe, Band 308, Paderborn (2013)
34. Koller, R., Kastrup, N.: Prinziplösungen zur Konstruktion technischer Produkte. Springer, Heidelberg (1998)
35. Langlotz, G.: Ein Beitrag zur Funktionsstrukturentwicklung innovativer Produkte. Ph.D. thesis, Institut für Rechneranwendung in Planung und Konstruktion, Universität Karlsruhe, Shaker Verlag, Band 2, Aachen (2000)
36. Langseth, H., Portinale, L.: Bayesian Networks in Reliability. In: Reliability Engineering & System Safety (2007)
37. Lee, J., Ni, D., Djurdjanovic, H., Qiu, H., Liao, H.: Intelligent Prognostic Tools and E-maintenance. Computers in Industry 57(6), 476–489 (2006)
38. Lindemann, U., Maurer, M.: Individualisierte Produkte - Komplexität beherrschen in Entwicklung und Produktion. Springer, Heidelberg (2006)

39. Blackenfelt, M.: On the Development of Modular Mechatronic Products. Royal Institute of Technology, KTH Stockholm (1999)
40. Mehrabian, A., Russell, J.A.: An Approach to Environmental Psychology. MIT Press, Cambridge (1974)
41. Mitchell, R.K., Agle, B.R.: Towards a Theory of Stakeholder Identification and Salience - Defending the Principle of Who and What Really Counts. Journal = Academy of Management Review 22(4), 11–14 (1997)
42. Nordsiek, D., Gausemeier, J., Lanza, G., Peters, S.: Early Evaluation of Manufacturing Costs within an Integrative Design of Product and Production System. In: Proceedings of the APMS 2010, International Conference on Advances in Production Management Systems, Como (2010)
43. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Engineering Design - A Systematic Approach, 3rd edn. Springer, Heidelberg (2007)
44. Pilone, D., Pitman, N.: UML 2.0 in a Nutshell: A Desktop Quick Reference. O'Reilly (2005)
45. Pook, S.: Eine Methode zum Entwurf von Zielsystemen selbstoptimierender mechatronischer Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 296, Paderborn (2011)
46. RailCab - Neue Bahntechnik Paderborn: The Project Web Site, `http://railcab.de` (accessed March 5, 2012)
47. Roth, K.: Konstruieren mit Konstruktionskatalogen: 1. Band: Konstruktionslehre, 3rd edn. Springer, Heidelberg (2000)
48. Sauer, T.: Ein Konzept zur Nutzung von Lösungsobjekten für die Produktentwicklung in Lern- und Anwendungssystemen. Ph.D. thesis, TU Darmstadt, VDI-Verlag, Düsseldorf (2006)
49. Scheffler, M.: Cost vs. Quality Trade-off for Gigh-density Packaging of Electronic Systems. Ph.D. thesis, Swiss Ferderal Institute for Technology, Eidgenössische Technische Hochschule, Zürich (2001)
50. Schmidt, A.: Wirkmuster zur Selbstoptimierung - Konstrukte für den Entwurf selbstoptimierender Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 204, Paderborn (2006)
51. Sondermann-Wölke, C., Geisler, J., Sextro, W.: Increasing the Reliability of a Self-optimizing Railway Guidance System (2010)
52. Stahl, T., Voelter, M.: Model-driven Software Development: Technology, Engineering, Management. John Wiley & Sons, Hoboken (2006)
53. Steffen, D.: Ein Verfahren zur Produktstrukturierung für fortgeschrittene mechatronische Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 207, Paderborn (2006)
54. Strube, G.: Wörterbuch der Kognitionswissenschaft. Klett-Cotta, Stuttgart (1996)
55. Thommen, J.P.: Managementorientierte Betriebswirtschaftslehre, 8th edn. Versus Verlag, Zürich (2008)
56. Tumer, I., Stone, R., Bell, D.: Requirements for a Failure Mode Taxonomy for Use in Conceptual Design. In: Proceedings of the International Conference on Engineering Design, Stockholm (2003)
57. Vaßholz, M., Gausemeier, J.: Cost-Benefit Analysis - Requirements for the Evaluation of Self-Optimizing Systems. In: Proceedings of the 1st Joint International Symposium on System-Integrated Intelligence, Hannover (2012)
58. Wilkinson, P., Kelly, T.: Functional Hazard Analysis for Highly Integrated Aerospace Systems. In: Proceedings of the Ground/Air Systems Seminar (1998)

# Chapter 5
# Methods for the Design and Development

Harald Anacker, Michael Dellnitz, Kathrin Flaßkamp, Stefan Groesbrink, Philip Hartmann, Christian Heinzemann, Christian Horenkamp, Bernd Kleinjohann, Lisa Kleinjohann, Sebastian Korf, Martin Krüger, Wolfgang Müller, Sina Ober-Blöbaum, Simon Oberthür, Mario Porrmann, Claudia Priesterjahn, Rafael Radkowski, Christoph Rasche, Jan Rieke, Maik Ringkamp, Katharina Stahl, Dominik Steenken, Jörg Stöcklein, Robert Timmermann, Ansgar Trächtler, Katrin Witting, Tao Xie, and Steffen Ziegert

**Abstract.** After the domain-spanning conceptual design, engineers from different domains work in parallel and apply their domain-specific methods and modeling languages to design the system. Vital for the successful design, are system optimization methods and the design of the reconfiguration behavior. The former methods enable the parametric adaption of the system's behavior, e.g. an adaption of controller parameters, according to a current selection of the system's objectives. The latter realizes structural adaption of the system's behavior, e.g. the exchange of software or hardware parts. Altogether, this leads to a complex system behavior that is hard to overview. In addition, self-optimizing systems are used in safety-critical environments. Consequently, the system's safety-critical behavior has to undergo a rigorous verification and testing process. Existing design methods do not address all of these challenges together. Indeed, a combination of established design methods for traditional technical systems with novel methods that focus on these challenges is necessary. In this chapter, we will focus on such new methods. We will introduce new system optimization and design methods to develop reconfigurations of the software and the microelectronics. In order to ensure the correctness of safety-critical functionality, we propose new testing methods and formal methods to ensure safety-properties of the software. We show how to apply virtual prototyping to deal with the complexity of self-optimizing systems and perform an early analysis of the overall system. As each domain applies its own modeling languages, the result of these methods are several overlapping models. In order to keep these domain-specific models consistent among all domains, we will introduce a

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*, Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_5, © Springer-Verlag Berlin Heidelberg 2014

new semi-automatic model synchronization technique. Each of these design methods are integrated with the reference process for the development of self-optimizing systems.

The principle solution forms the basis of the design and development. Engineers of the involved domains derive their domain-specific models from the it. This is, however, an error-prone and tedious task. Therefore, we will introduce a semi-automatic model transformation techniques (cf. Sect. 5.1) that enables engineers to, e.g. derive an initial controller hierarchy or an initial software architecture. Afterwards, each domain details these models. This may involve changes that have an impact on the other domains. In order to keep the models of all domains consistent, we will propose a model synchronization technique (cf. Sect. 5.1.3).

The system must consider several concurrent objectives in different Application Scenarios.

This requires methods for optimizing the system with respect to these objectives and appropriate adaption methods. System optimization methods origin from the research areas of applied mathematics and artificial intelligence. The methods determine the optimal system behavior or a set of optimal compromises for several concurrent objectives. Practically, this is a formalism to compute optimal controller parameters or optimal configurations of the system structure (cf. Sect. 5.3). Then, it is the task of engineers from the domains mechanical, electrical/electronic, control, and software engineering to specify the corresponding change of the system's behavior, i.e. the reconfiguration of the system.

The system can perform reconfigurations on every system level (cf. Sect. 1.4.3). In particular, this requires new design methods for the application software, the system software, and the hardware modules to specify reconfiguration. Furthermore, reconfiguration is often safety-critical and must fulfill hard real-time constraints. Consider the RailCab's reconfiguration behavior to build a convoy as an example (cf. Sect. 2.1.7): The RailCab must reconfigure the controller behavior to consider the distance to the preceding RailCab if the RailCab joins a convoy as a member. In fact, if this function is not free from design faults or the system cannot execute the reconfiguration within a certain time, a crash may happen. Therefore testing and formal verification methods are crucial to ensure the safety of the system's complex behavior and its real-time properties.

On the level of the application software, software engineers specify the communication behavior and the switching between alternative behavior implementations. We apply a component-based design method called MECHATRONICUML that considers hard-real time constraints for the communication behavior, the reconfiguration of controllers, and the reconfiguration of software components. In MECHATRONICUML, formal verification techniques are applied to ensure safety constraints and the real-time properties of the system.

As a consequence of reconfigurations of the application software, the software's resource and performance demands changes. Usually, the system must reconfigure hardware modules to meet the changed requirements of the application software again. For instance, a change of the communication behavior may require a change of the physical communication topology or the implementation of a communication protocol on the hardware to meet the performance requirements. Different hardware techniques such as FPGAs or multi-processor platforms are capable to realize these reconfigurations. We will present design methods, architectures, and modeling approaches to design dynamically reconfigurable hardware for different techniques and enable flexible and robust implementation of dynamically reconfigurable hardware (Sect 5.4). In particular, a layered architecture such as PALMERA (Paderborn Layer Model for Embedded Reconfigurable Architectures) can be applied to abstract from the different hardware techniques. Based on PALMERA the design-flow INDRA guides engineers through the different steps towards the realization of information processing systems based on dynamically reconfigurable hardware.

The system software forms the interface between the application software and the dynamic reconfigurable hardware. Hence, the system software must define a common interface to trigger changes of the hardware. Furthermore, it must adapt to changing available resources and changing resource demands while operating under hard real-time constraints. This requires new concepts and design methods for the system software. ORCOS (Organic Reconfigurable Operating System) is a real-time operating system that provides operating system services and an architecture to master these challenges (cf. Sect. 5.5). For instance, the FRM (flexible resource management) allows an overallocation of resources to optimize the resource availability under changing resource demands (cf. Sect. 5.5.2).

The result of the design and development is a complex composed behavior developed by different engineers. This leads to a specification, that is hard to overview. In addition, engineers must ensure the correctness of safety-critical functionality as early as possible during the design. One solution to cope with the complexity is to build and test a virtual prototype. Virtual prototyping enables engineers to perform experiments during early development phases. It requires models of the system that are often created by several tools. We will introduce a concept of a virtual environment and methods to extend the environment and integrate models of the domains involved.

This chapter is structured as follows: First, we will describe the model transformation techniques to derive the domain-specific models from the principle solution and the model synchronization technique to keep the domain-specific models consistent (cf. Sect. 5.1). In Sect. 5.2, we will introduce the design of the communication software and reconfiguration behavior with MECHATRONICUML. Novel system optimization approaches that origin from mathematics and artificial intelligence follow in Sect. 5.3. We will describe technologies and design methods for dynamic reconfigurable hardware in Sect. 5.4. In Sect 5.5, we will focus on the system software and introduce the self-optimizing real-time operating system ORCOS. Finally, we will introduce virtual prototyping and advanced testing methods in Sect. 5.6.

## 5.1 Automatic Model Transformation and Synchronization

Jan Rieke

In the domain-spanning conceptual design, experts from all domains have elaborated the principle solution. This principle solution covers all domain-spanning relevant information, i.e. all interfaces and overlaps between different domains are described in this model. Thus, the principle solution can serve as a starting point for the domain-specific design and development.

In this section, we will show how *model transformation techniques* can be applied to *automatically derive initial domain-specific models* that are consistent with the principle solution and all other domain-specific models. Basically, these initial models contain skeletons that are filled by the domain engineers in the design and development phase. We will explain a model transformation that generates software engineering models from a principle solution in Sect. 5.1.2.1. In Sect. 5.1.2.2, we will also show how initial control engineering models can be generated.

Ideally, the principle solution covers all domain-spanning aspects. Thus, there should be no need for further domain-spanning coordination. However, in practice, the principle solution rarely captures every domain-spanning concern. Additionally, changes to the overall system design may become necessary later on, e.g. due to changing requirements. Therefore, cross-domain changes may become necessary during the domain-specific design and development. Sect. 5.1.3 explains in detail how *model synchronization techniques* can be applied in such a scenario.

Before describing the model transformation and synchronization technique in detail, let us have a closer look at an example.

### *5.1.1 Example Scenario*

As a running example, let us consider the RailCab system (cf. Sect. 2.1). When driving in a convoy, all RailCabs (except for the convoy leader) control their velocity based on the distance to the RailCab traveling in front of them. To measure the distance, a distance sensor is mounted in the front of each RailCab. Figure 5.1 shows how the different models evolve in the exemplary scenario described below.

To illustrate transformations and synchronizations that may become necessary throughout the development, assume the following exemplary process. After the system engineers design the principle solution, we apply model transformations to the different domain-specific models (step 1). The control engineers then start implementing the controllers (step 2). In particular, they elaborate on the velocity control strategies for driving in a convoy as a follower based upon the distance measured by the distance sensor. This is a domain-specific refinement that has no influence on the models of other disciplines.

Modern mechatronic systems incorporate self-healing to repair the system in case of failure. As described in Sect. 5.2.7, the software engineers perform an analysis of the self-healing operations in order to determine whether they reduce the probability of hazards successfully. In our example, the distance sensor could fail or send bad

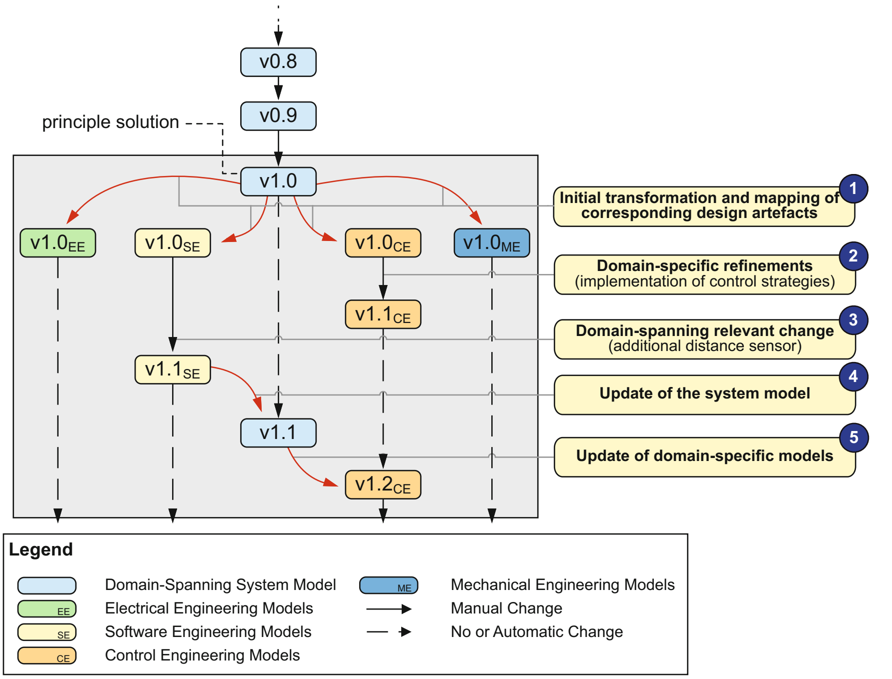

v0.8
v0.9
principle solution
v1.0
v1.0EE
v1.0SE
v1.0CE
v1.0ME
v1.1CE
v1.1SE
v1.1
v1.2CE
1 Initial transformation and mapping of corresponding design artefacts
2 Domain-specific refinements (implementation of control strategies)
3 Domain-spanning relevant change (additional distance sensor)
4 Update of the system model
5 Update of domain-specific models
Legend
Domain-Spanning System Model
EE Electrical Engineering Models
SE Software Engineering Models
CE Control Engineering Models
ME Mechanical Engineering Models
Manual Change
No or Automatic Change

**Fig. 5.1** Evolution of different models during the development process

data. It may turn out that even with self-healing the hazard probability can not be reduced to an acceptable level: The hazard of two RailCabs colliding during convoy mode due to a failing distance sensor exceeds the acceptable hazard probability of the system. Thus, software engineers should propose adding redundancy by adding a second distance sensor. They add a new sensor measurement component to their software model (step 3). This is a domain-spanning relevant change, i.e. it affects the domain-spanning system model as well as several domain-specific models. In particular, the velocity control strategy must be modified.

Thus, we use **model synchronization techniques** to propagate the change to the system model (step 4). In contrast to model transformation, which translates complete models, the idea of model synchronization is to modify only the model elements that have been changed after the initial model transformation. Thus, model synchronization is also called an *incremental update*. Version 1.1 of the system model now contains a second distance sensor. To allow all engineers to react to change, it is propagated further to all affected domain-specific models. For instance, the control engineering model is updated, again using model synchronization techniques (step 5).

The control engineers can now modify their control strategy to use both sensor data as input. In step 5, it is crucial that the domain-specific model is updated in a way, so that all refinements and implementations that have been added to it in the meantime (see step 2) are retained.

Next, we will describe in detail a) how to derive initial domain-specific models, and b) how to propagate changes to the system model and further on to domain-specific ones.

## *5.1.2 Deriving Initial Domain-Specific Models from the System Model*

In this section, we will present two example transformations from the system model to domain-specific models. We will use the principle solution as the input to these transformations to create models for the domain-specific design and development. However, these transformations can also already be used during the conceptual design to generate domain-specific models, for instance, for early simulation and verification.

First, we will describe how the active structure can be used to derive initial software component models in MECHATRONICUML[9], and how the Behavior–States model is transformed to an initial software statechart. Next, we will show how control engineering models (MATLAB/Simulink and Stateflow) can be derived.

### 5.1.2.1 Transformation from CONSENS to Software Engineering Models

Figure 5.2 shows the basic principles of the transformation from CONSENS to MECHATRONICUML software models. In the active structure, you can see small colored annotations above the system elements. These so-called *relevance annotations* define which element is relevant to which domain-specific model. For instance, “SE” and “CE” denotes software and control engineering, respectively.

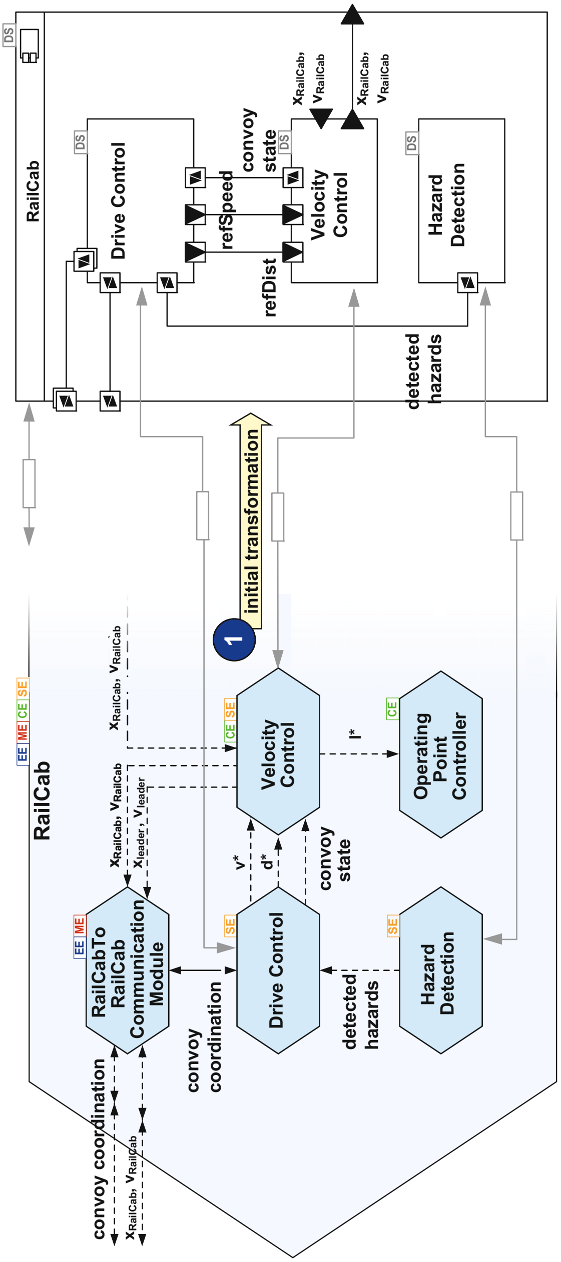
The central idea of mapping is that every system element that has a software engineering relevance annotation (i.e. it fulfills software functions) should be represented by a software component in the MECHATRONICUML model. The information flows between system elements are mapped to ports and connectors in MECHATRONICUML.

Generally, we distinguish between continuous and discrete components. *Continuous components* are typically controllers that continuously process input data from sensors to compute outputs for actuators Typically, control engineers implement them. However, MECHATRONICUML allows integrating them as continuous components. **Continuous components** are black-box components, i.e. no actual behavior is attached to continuous components in MECHATRONICUML. In this way, they define the interface to control engineering in a MECHATRONICUML software model.

In contrast, the behavior of *discrete components* is implemented using MECHATRONICUML (cf. Sect. 5.2). **Discrete components** communicate with each other via discrete ports using asynchronous, message-based communication defined in

[9] See Sect. 5.2 for a detailed explanation of MECHATRONICUML.

**Fig. 5.2** Initial transformation from the active structure to a software component diagram (adapted from [70])



real-time statecharts. Discrete components can also send or receive signals to or from continuous components using hybrid ports. In Fig. 5.2, the components *Drive Control* and *Velocity Control* have both discrete and continuous ports.

The transformation creates discrete components in MECHATRONICUML for every system element that is relevant for software engineering. We use a technique called *Triple Graph Grammars* (TGG) for defining the model transformations to the different domain-specific models. TGGs are a graph-based, declarative technique to define mappings between two models, invented by Schürr (1994) [188].

Figure 5.3 shows a TGG rule that is part of the TGG rule set that implements this mapping.

A TGG rule describes which model elements in one or more source models relate to which model elements in one or more target models. In Fig. 5.3, the two source models are the two left columns, *AS Annotation* and *active structure*.[10] The target models are located in the right columns, *Component Diagram* and *UML Annotation*. In the middle column, the so-called *correspondence model* is described, which is a kind of trace model, storing relations between the models. It is used to identify corresponding model parts when incrementally updating models. The green parts of the rule, additionally marked with "++"s, is the actual mapping, stating that a *SystemElementInstance* that has a *Relevance* annotation must be mapped to *Property*, also with a *Relevance* annotation. The b/w part of the rule is the *context*, defining in which situations the mapping must be valid.

We use our TGG Interpreter Tool Suite [203] to define and automatically execute these TGG rules. Given a domain-spanning system model and a TGG rule set, the TGG Interpreter can automatically create the corresponding domain-specific models. We refer to Greenyer and Kindler (2010) [78] for further details on TGGs. Gausemeier et al. (2009) [70] describe the principles of the transformation from the active structure to software models in detail.

#### 5.1.2.2 Transformation from CONSENS to Control Engineering Models

Figure 5.4 shows the basic principles of the transformation from CONSENS to MATLAB/Simulink control engineering models.

Generally, every system element that is relevant to software or control engineering is mapped to a Simulink block. The system elements relevant only to software engineering, however, are just placeholders. When the software engineers finish the actual implementation, this implementation is inserted. This is because we use a MATLAB/Simulink model at the end of the development process as a combined software/control engineering model from which code is generated. Thus, all artifacts of the software engineering domain are integrated into this MATLAB/Simulink model.

To allow the integration of discrete software components that use asynchronous, message-based communication and reconfiguration, we use a message bus

[10] Due to technical reasons (e.g. to allow easy extensibility), annotations are stored in two separate models. Thus, they are located in the separate columns *AS Annotation* and *UML Annotation* in Fig. 5.2.

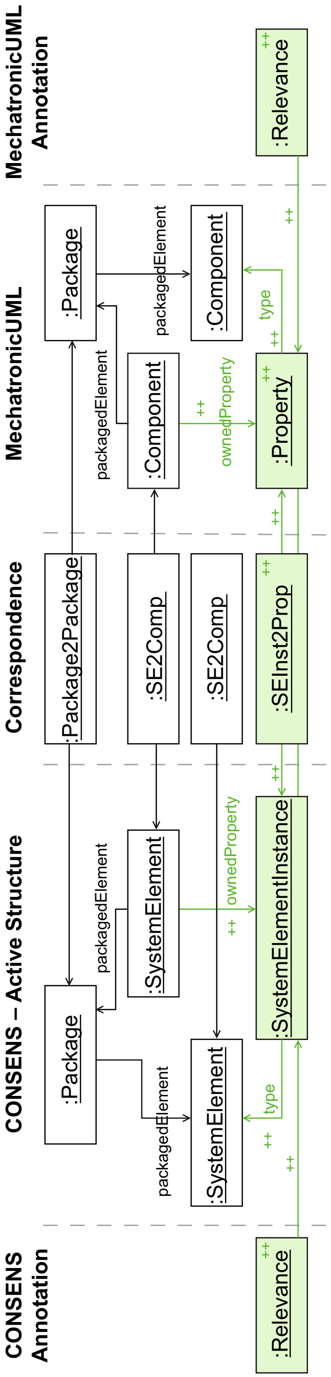


**Fig. 5.3** TGG rule for mapping system elements to software components (adapted from [70])

**Fig. 5.4** Initial transformation from the active structure to a MATLAB/Simulink control engineering model

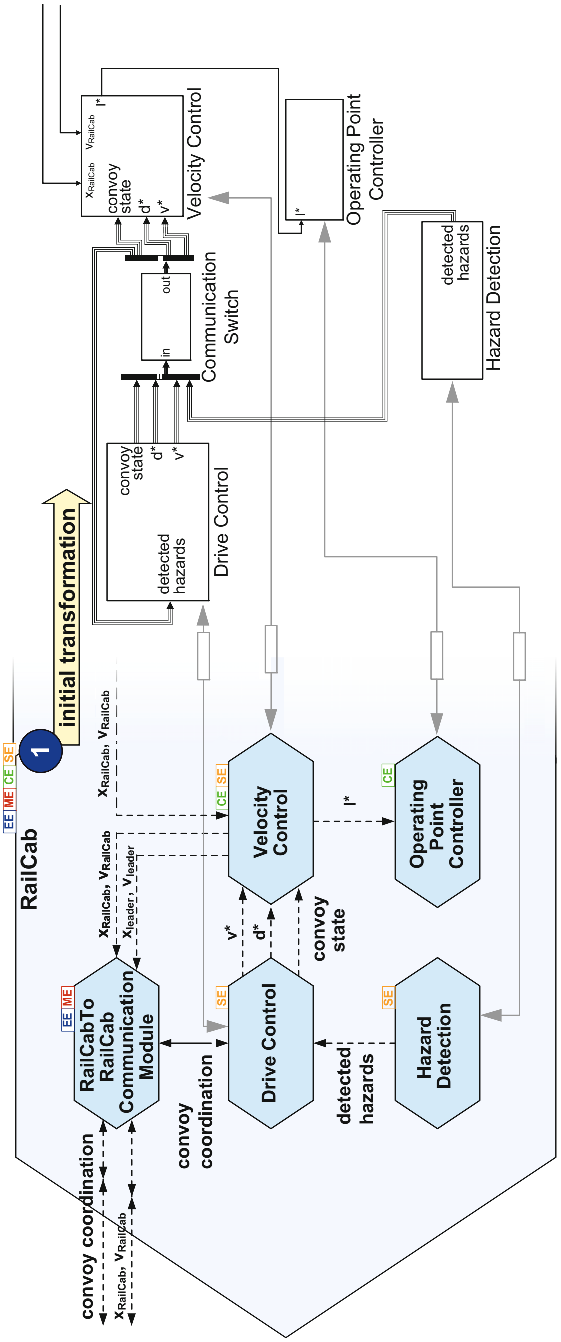
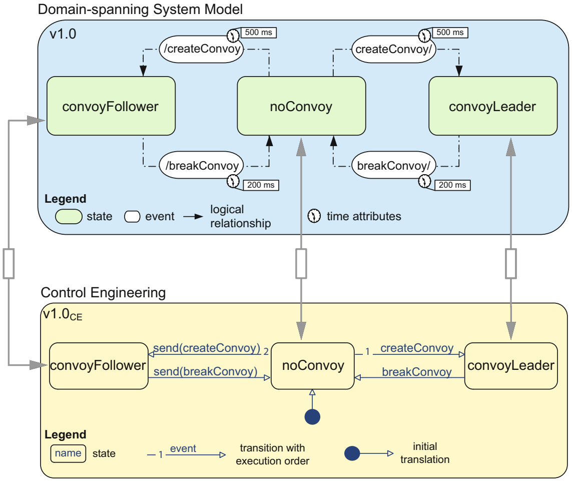


**Fig. 5.5** Initial transformation from Behavior–States to a MATLAB/Stateflow model



approach. The communication between two discrete components is implemented using a *Communication Switch*. This switch connects every component and is responsible for forwarding sent messages to the correct recipient. This is necessary to allow changing communication structures as required when reconfiguring a system. Signal-based information flow, like the $I^*$ value signal from *Velocity Control* to *Operating Point Controller*, is mapped to connected outputs and inputs of the respective blocks.

Furthermore, we use behavioral models of the principle solution to generate MATLAB/Stateflow control engineering models. Figure 5.5 shows such a transformation of behavioral models.

Rieke et al. (2012) [180] describe the principles of the transformation of state-based models. Heinzemann et al. (2012) [93] give technical details on the generation of MATLAB/Stateflow and Simulink models.

### 5.1.3 *Synchronizing Models during the Domain-Specific Refinement Phase*

Although most domain-spanning relevant information should already be present at the end of the conceptual design, changes to the system in development may become necessary during the domain-specific design and development. For instance, requirements may still change during later phases, or it may turn out that some aspect of the system must be implemented in another way. This easily leads to changes that affect both the domain-spanning system model and several domain-specific models. Furthermore, engineers may have already generated early domain-specific models during conceptual design, to allow early checks and simulations of different concepts and ideas. It is reasonable to keep these early models and to reuse and refine them during the design and development.

This requires keeping the development models consistent during all phases of the development. Manually checking and restoring the consistency of all models is a time-consuming and error-prone task. Therefore, we apply similar methods as with the derivation of initial models (described in the previous section) to synchronize models during the development.

First, we will describe how the system model is updated when changes in a domain-specific model occur. Next, we will show how domain-specific models can be updated with respect to these system model changes.

#### 5.1.3.1 Updating the System Model

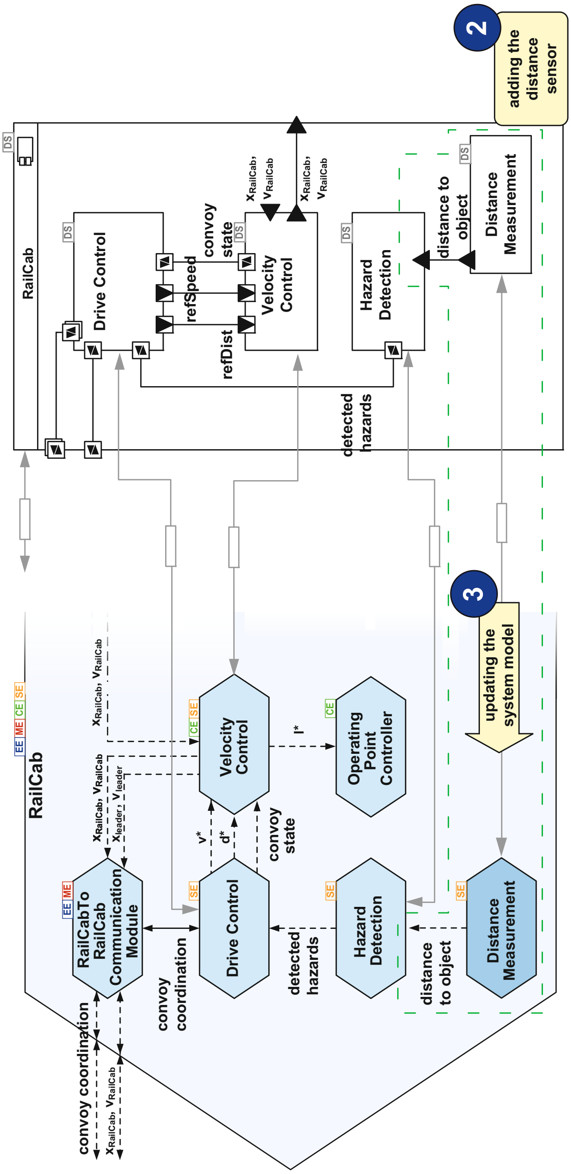
As described in Sect. 5.1.1, an extra distance measurement component is added to the software model (step 3 in Fig. 5.1). Our model transformation approach forwards this change to the system model (step 4 in Fig. 5.1). Figure 5.6 shows the result of this step.

We again use TGGs to perform such model synchronization operations. TGG rules can be applied bidirectionally, i.e. transformation and synchronization operations can be performed both from the system model to the software model and vice versa. Here, we apply the TGG rules reverse, propagating the change from the software model to the system model. The added *Distance Measurement* system element is shown on the left side of Fig. 5.6.

We do not want to simply run the transformation again in backwards direction, as this would completely re-create one model. Thus, the core idea is to only update modified model parts and leave everything else untouched. For every model element, we check whether mapping of this element is still valid. To do so, our approach uses the existing trace information that is stored inside the correspondence graph. Using this correspondence graph, it can identify corresponding model elements in the two models and then check the consistency of these model elements by testing whether the TGG rule that was applied there still holds. The approach only modifies a model element if a rule does not hold any more relevance. Such an approach is called *incremental model transformation* or *model synchronization*.

We have developed a new, improved model synchronization algorithm that is tailored for the use in mechatronic system design. More specifically, it prevents the loss of information in models during the synchronization process. This is especially required when synchronizing development models of mechatronic system, as these models have different abstraction levels and/or different views: The system model is usually more abstract than the domain-specific models that contain concrete implementation details. Thus, the domain-specific models may contain information that is not part of the system model. For instance, the Stateflow model shown in the lower part of Fig. 5.5 is later refined such that it contains details of controller reconfigurations that happen when switching convoy states. Thus, the Stateflow model now contains information that is not present in the abstract system model. When the system model is changed, this change may affect parts of the Stateflow model that has been refined. Our synchronization algorithm avoids affecting these

**Fig. 5.6** Updating the active structure using the altered software component diagram (from [70])



refinements when updating this domain-specific model. Rieke et al. (2012) describe such a change scenario in detail [180]. For details on the improved model synchronization algorithm, see Greenyer et al. (2011) [79].

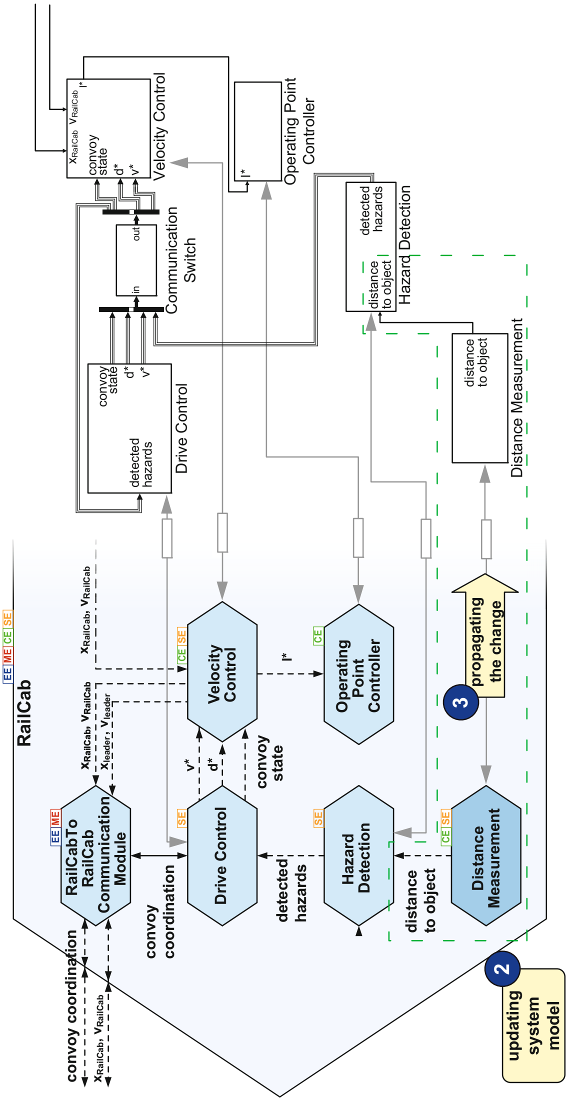


**Fig. 5.7** Updating the MATLAB/Simulink control engineering model using the updated active structure diagram

### 5.1.3.2 Updating Control Engineering Models

After updating the system model, these changes must be propagated to other affected domain-specific models (step 5 in Fig. 5.1). Figure 5.7 shows how the added *Distance Measurement* system element can also be added to the control engineering model.

This is again achieved by rerunning the transformation incrementally, leaving the unaffected parts untouched and only adding a new block with its respective inputs, outputs and lines.

When changes to a model occur, we are able to update other affected models automatically in most cases, using these improved model transformation and synchronization techniques. However, there might be cases where user decisions are indispensable, for instance when there are different possibilities to propagate a specific change. Thus, it is reasonable to combine this technique with means for user interaction [79].

## 5.2 Software Design

Christian Heinzemann, Claudia Priesterjahn, Dominik Steenken, and Steffen Ziegert

Self-optimizing mechatronic systems execute a great amount of software to coordinate the operations of the system. In the following, we will refer to that software as the *discrete* software of the system as opposed to the controller software. The RailCab demonstrator for example (cf. Sect. 2.1) needs discrete software to manage the necessary communication for getting admission to drive onto a track section and, especially, for driving in convoy mode. In convoy mode, RailCabs need to execute complex coordination behavior for maintaining the convoy when the convoy consists of more than two RailCabs. Since RailCabs can join or leave a convoy during a journey, a flexible structure for the specification of the coordination is needed. The required small distances between RailCabs in a convoy imply real-time coordination between the speed control units of the RailCabs. This is safety-critical and requires the software engineer to address a number of constraints when designing the RailCabs' control software.

In the design and development, the software engineers apply the MECHATRONICUML method [53, 75] for designing the discrete software of mechatronic systems, especially of self-optimizing mechatronic systems (cf. Sect. 3.3.3). **MECHATRONICUML** enables a component-based specification of the discrete software with a special focus on specifying the communication and reconfiguration behavior of a self-optimizing mechatronic system. The development process for developing with MECHATRONICUML in the course of the design and development is shown in Fig. 3.11 on Page 84. We illustrate the development with MECHATRONICUML by providing an overview of the general concepts of MECHATRONICUML [1, 43, 75] and recent extensions [53, 54, 91, 196] in the course of this section. The complete, technical language specification of MECHATRONICUML can be found in [18].

The software engineers start the development with MECHATRONICUML by deriving a component model for the discrete software as discussed in Sect. 5.2.1. In the next step, the communication requirements need to be decomposed based on the components of the component model as described in Sect. 5.2.2. The communication protocols that define the message-based communication of the components are specified formally by using real-time coordination patterns and verified with our design-time verification procedure as explained in Sect. 5.2.3. Afterwards, the component's discrete communication behavior is specified as described in Sect. 5.2.4.

In Sect. 5.2.5 we will outline how the complete hybrid system is simulated. When the simulation is successful, the deployment of software components to hardware is specified as explained in Sect. 5.2.6. Finally, we will outline an analysis of self-healing operations in Sect. 5.2.7 and the code generation in Sect. 5.2.8.

## *5.2.1 Component Model*

The software development with MECHATRONICUML starts by deriving an initial component model for the system, because MECHATRONICUML follows the component-based approach [198] for developing software. Each component encapsulates part of the system functionality and the components only interact via well-defined interfaces, called ports. An initial component model is derived from the Active Structure by using the transformation presented in Sect. 5.1.2.1. Since the Active Structure only contains components that affect more than one discipline, it might be necessary to refine the component model by splitting the behavior of a component into several subcomponents. That reduces the complexity of the single components which, in turn, enables the reuse of existing components and makes their verification more efficient.

Fig. 5.8 shows the *DriveControl* component of the RailCab that has been derived from the system element *DriveControl* as shown in Fig. 5.2. The *DriveControl* component encapsulates the software controlling the driving operations of the RailCab. In our example, a RailCab will either be a coordinator or a member of a convoy, but not both at the same time. Therefore, the developer may decide to split the behavior of the RailCab component into subcomponents. The two components *ConvoyCoordination* and *MemberControl* encapsulate the behavior of being coordinator and of being member respectively. In addition to these components, each RailCab requires a component *SpeedControl* which defines the speed for the RailCab which serves as the reference speed for the controller. If the RailCab is a convoy member, the reference speed and an additional reference distance to the preceding RailCab in the convoy are received by the *MemberControl* and propagated by *SpeedControl* to the controller.

In self-optimizing mechatronic systems, the components interact by means of message passing via their ports. In MECHATRONICUML, the behavior that defines an interaction between two components is specified by so-called real-time coordination patterns (cf. Sect. 5.2.3). In Fig. 5.8, the *DriveControl* interacts with other components using the ports *coordinator*, *member*, *hazardReceiver*, *convoyState*, *refSpeed*, and *refDist*. The former four ports are discrete ports that execute a state-based communication protocol specified by a real-time coordination pattern (cf. Sect. 5.2.3). The *refSpeed* and *refDist* ports are so-called hybrid ports which are used for providing a value, in this example the reference speed and reference distance for the RailCab, to a controller.

The behavior of components and ports is defined using a state-based approach called **real-time statecharts** (RTSC). RTSCs are a combination of UML

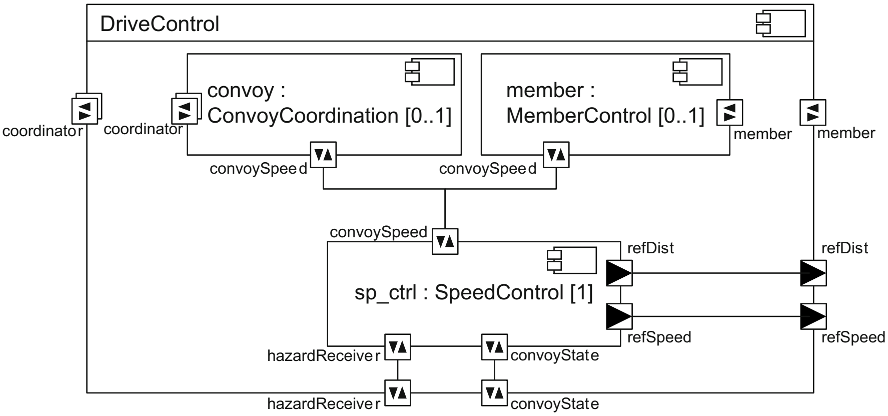


**Fig. 5.8** *DriveControl* component of the RailCab

statemachines and timed automata [8]. We will provide more information on RTSCs using an example in Sect. 5.2.3.

The MECHATRONICUML component model distinguishes between components and component instances. A component instance is the occurrence of a component in a system. Component instances are connected via their ports for specifying a concrete system architecture, called component instance configuration.

Fig. 5.9 shows a component instance configuration that consists of three instances of the component *RailCab* (cf. Fig. 5.2). The *RailCabs* drive in a convoy because they execute the real-time coordination pattern *ConvoyCoordination* which we will introduce in detail in Sect. 5.2.3.

### *5.2.2 Decompose Communication Requirements*

In a self-optimizing mechatronic system, the single components often interact and exchange different kinds of data. In the example in Fig. 5.9, RailCabs interact with each other for two reasons. First, they communicate to coordinate the convoy drive and, second, a RailCab needs to transmit its current position to adjacent RailCabs in the convoy for controlling the distance. The communication protocols defining the necessary message exchange are specified by real-time coordination patterns of MECHATRONICUML. A developer should specify one real-time coordination pattern for each reason for interaction to achieve separation of concerns. This in return will reduce the complexity of the single real-time coordination patterns, allow a more efficient verification, and enable their reuse in different systems.

The requirements for the real-time coordination patterns are specified by means of **Modal Sequence Diagrams** (MSDs) as described in Sect. 4.3. The MSD specification, however, does not distinguish the different communication protocols. Therefore, the developer needs to decompose the MSDs according to the communication protocols that are needed in the system. For the example in Fig. 5.9, we obtain

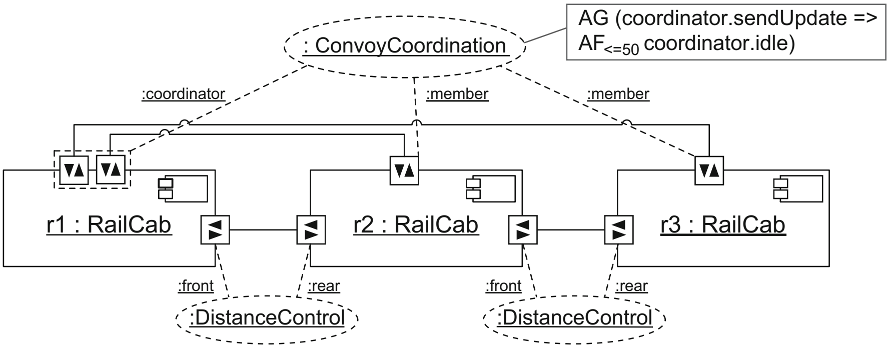


**Fig. 5.9** Component instance configuration of a convoy with three RailCabs

one set of MSDs for the *ConvoyCoordination* and one set of MSDs for the *DistanceControl*. Then, the developer needs to define a real-time coordination pattern as described in Sect. 5.2.3 for each of the communication protocols. These real-time coordination patterns are then associated with the ports and connectors of the component model as shown in Fig. 5.9.

In addition, the developer may split components into several subcomponents as illustrated in the *DriveControl* component in Fig. 5.8. In this case, the interactions need to be associated with subcomponents that will implement the interaction. This step might require a further derivation of MSDs that define the requirements for the communication within a component. In the *DriveControl* component, the developer needs to specify MSDs for the interaction of *SpeedControl* with *ConvoyCoordination* and *MemberControl*.

### *5.2.3 Real-Time Coordination Patterns*

The communication behavior of the components is specified formally by using real-time coordination patterns. The developer needs to specify a real-time coordination pattern for each connector between components in the component model. In MECHATRONICUML, real-time coordination patterns are specified independent of a concrete component to allow reusing them in different systems. Thus, the developer either needs to specify a new real-time coordination pattern based on the communication requirements as described in Sect. 5.2.3.1 or he may reuse an existing real-time coordination pattern. The real-time coordination pattern is refined to the specific components as part of process step "Specify Discrete Behavior" (cf. Sect. 5.2.4).

The communication behavior is safety-critical. In our example, errors in the communication between convoy coordinator and convoy members may lead to an accident. If a RailCab still operates in convoy mode while the convoy coordinator assumes that it has left the convoy, a crash may occur if the convoy brakes because

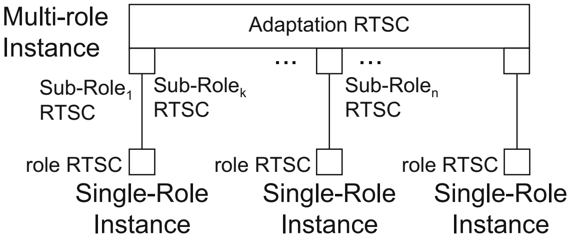


**Fig. 5.10** Instance of a real-time coordination pattern with a multirole

the RailCab will not be notified. We can prove the correctness of the communication behavior by using our design-time verification procedure outlined in Sect. 5.2.3.2.

### 5.2.3.1 Specification of Real-Time Coordination Patterns

A real-time coordination pattern defines the required communication between two communications partners independent of a concrete component implementation. We call the communication partners *roles*. In this section, we focus on 1:*n* communication where one role communicates with *n* other roles all executing the same behavior [53]. In a RailCab convoy (cf. Fig. 5.9), one RailCab serves as a coordinator and needs to communicate with the *n* other members of the convoy. The coordinator is required, e.g. for defining a reference speed for the whole convoy and to coordinate acceleration and braking maneuvers.

RailCab *r1* is the coordinator of the convoy, while *r2* and *r3* are members. Therefore, *r1* executes an instance of the *coordinator* role of the *ConvoyCoordination* real-time coordination pattern. RailCabs *r2* and *r3* execute an instance of the *member* role. The instances of the *DistanceControl* real-time coordination pattern are used for controlling the distance between two successive RailCabs in a convoy.

Since the *coordinator* role instance communicates with *n member* role instances, we call it a *multirole*. The *member* role instance, which communicate with only one *coordinator* role instance is called a *singlerole*. Fig. 5.10 shows the general structure of an instance of a real-time coordination pattern with a multirole instance.

The multirole instance consists of an adaptation real-time statechart and *n subrole* real-time statecharts. Each of the subrole instances manages the communication with exactly one singlerole instance. The adaptation real-time statechart is responsible for creating and deleting subrole instances, e.g. if RailCabs join or leave a convoy. In addition, the adaptation real-time statechart is used to coordinate the subrole real-time statecharts, e.g. to trigger that they send data to the member RailCabs in a defined order.

Fig. 5.11 shows the real-time statechart that defines the behavior of the singlerole *member*. The real-time statechart starts its execution in the initial state *waitUpdate*. It waits for 500 time units for an *update* message to arrive. Messages are sent asynchronously between different roles, i.e. the receiver stores the message in a buffer and may process it at a later point in time. If the message arrives in time, the real-time statechart switches to *sendAck* thereby resetting the clock *c* to 0. If the message

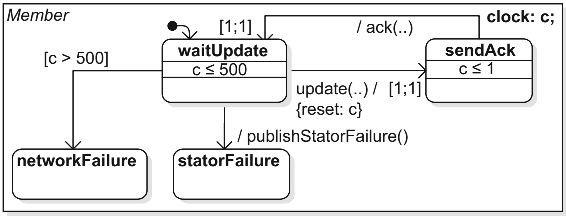


**Fig. 5.11** Real-time statechart of a convoy member

does not arrive in time, it switches to state *networkFailure*. The state *sendAck* is left after 1 time unit by sending a message *ack* and switching to *waitUpdate*.

Fig. 5.12 shows the real-time statechart that defines the behavior of the multirole *coordinator*. The real-time statechart of a multirole always consists of one state that contains two parallel regions, which is *Coordinator_Main* in the example. One region contains the adaptation real-time statechart while the other contains the subrole real-time statechart that is executed by all subrole instances. At run-time, we obtain one real-time statechart instance of the subrole real-time statechart for each subrole instance.

The *coordinator* subrole real-time statechart in the lower region of *Coordinator_Main* is the pendant to the *member* real-time statechart of Fig. 5.11. It is initially in state *idle*. The transition from *idle* to *sendUpdate* is triggered by a synchronous internal event *next* which is parameterized by an integer. Synchronous events cause

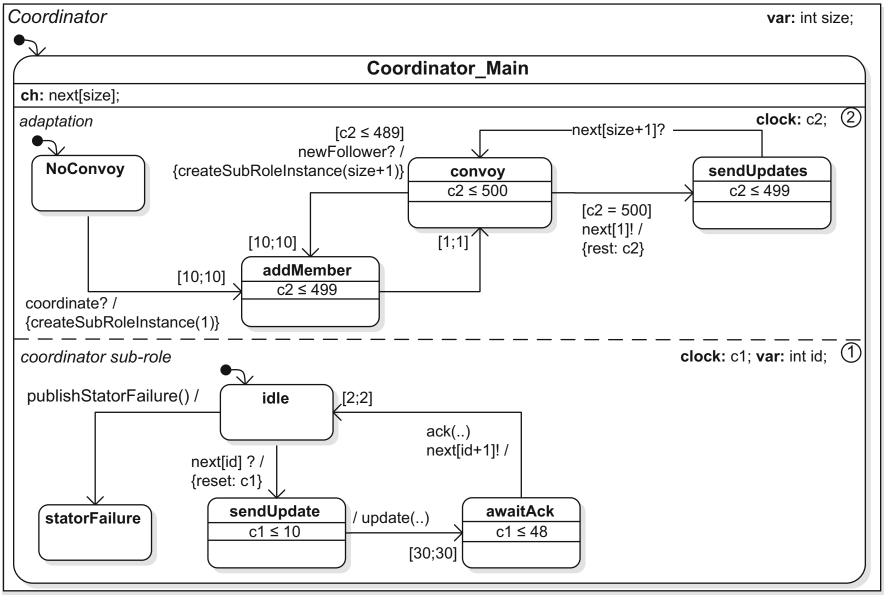


**Fig. 5.12** Real-time statechart of the convoy coordinator

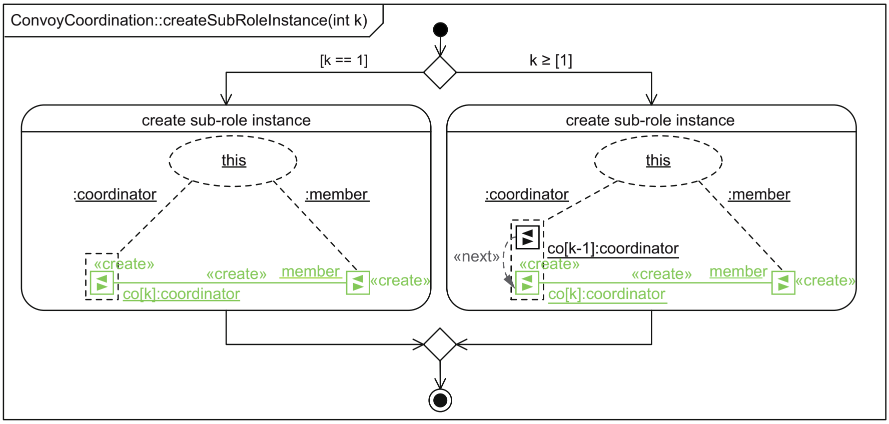


**Fig. 5.13** Component story diagram modeling the creation of a subrole instance

the sender transition (event suffixed by !) and the receiver transition (event suffixed by ?) to fire simultaneously. Additionally, sender and receiver must provide and expect the same integer parameter. In the state *sendUpdate*, the real-time statechart may spend up to 10 time units before it sends the *update* message and switches to the state *awaitAck*. Executing this transition takes a minimum and a maximum of 30 time units which is indicated by the deadline in square brackets. The transition from *awaitAck* to *idle* is triggered by the receipt of the *ack* message from the *member* role. It triggers the next subrole using the synchronous event *next*, incrementing the expected integer by 1.

The adaptation real-time statechart in the upper region of *Coordinator_Main* starts in state *noConvoy*. If the RailCab is chosen to coordinate the convoy, it is triggered by the synchronous event *coordinate* and switches to *addMember*. The side effect *createSubRoleInstance* at the transition triggers the component story diagram of Fig. 5.13 that creates a new subrole instance in the *coordinator* multirole instance. Then, the real-time statechart switches to the state *convoy*. In the state convoy, the real-time statechart triggers the first subrole instance every 500 time units using the synchronous event *next*. The transition from *sendUpdate* back to *convoy* synchronizes with the last subrole instance after it has successfully received the *ack* from the *member* role. Back in the state convoy, the real-time statechart can only be triggered by the synchronous event *newFollower* and switch to *addMember*. Again, the side effect *createSubRoleInstance* of the transition executes the component story diagram of Fig. 5.13 for creating a new subrole instance.

**Component story diagrams** [200] are a special kind of graph transformation rules [184] that use the concrete syntax of MECHATRONICUML. We use component story diagrams for specifying run-time reconfiguration operations, i.e. the creation and deletion of component instances and connections. The component story diagram of Fig. 5.13 instantiates a new connection to a new member that wants to join the convoy at position *k*. In the component story diagram, we distinguish between creating the first connection and creating further connections. In the activity node on the left, we create the first connection between coordinator and a member in the real-time coordination pattern. The *this*-variable represents the instance of the *ConvoyCoordination* real-time coordination pattern which called the component story diagram from the adaptation real-time statechart of its multirole instance. In addition, the multirole instance, modeled by the dashed rectangle, is bound. Then, the parts of the rule annotated with «create» are created and the connection is established. In the activity node on the right, the subrole instance with index $k-1$ is bound additionally and the new subrole instances is created as a successor to this subrole instance.

#### 5.2.3.2 Design-Time Verification of Real-Time Coordination Patterns

The correctness of software for self-optimizing mechatronic systems is often safety-critical, especially if the software influences the physical movement of the system. That requires the software to meet high quality standards to ensure its safe operation. Traditional testing-based development approaches are not able to guarantee functional correctness. Design-time verification, however, is a method to give a mathematical proof that a software is functionally correct with respect to a formal specification [13]. In this section, we will illustrate how design-time verification can be used to ensure that the communication within a self-optimizing mechatronic system modeled by real-time coordination patterns is safe.

The real-time coordination patterns used in self-optimizing mechatronic systems are often subject to run-time reconfiguration. An example is given by the *Convoy-Coordination* real-time coordination pattern introduced in Sect. 5.2.3. In such real-time coordination patterns, the behavior is defined by a syntactical combination of real-time statecharts and component story diagrams. Consequently, a verification procedure needs to take both into account.

Existing approaches and corresponding tools for design-time verification do not provide sufficient support for self-optimizing mechatronic systems that adhere to real-time constraints and use run-time reconfiguration. Graph-based tools like GROOVE are very effective for verifying untimed graph transformation systems (GTS) [113], but are still limited, especially with respect to verification of timing properties. Timed model checkers such as Kronos [31] or UPPAAL [21], which support the verification of real-time statecharts, provide no means for specifying dynamic object creation and deletion.

Our method for design-time verification combines the strengths of both approaches for verifying real-time coordination patterns with run-time reconfiguration. It is executed at design-time by the software engineer while creating the real-time coordination patterns as opposed to run-time verification which is performed while the mechatronic system is running [69, D.o.S.O.M.S. Sect. 3.2.14] .

Fig. 5.14 provides an overview of the single steps of our verification procedure. It requires two inputs: a real-time coordination pattern and a set of requirements that need to be verified. The textual requirements informally state the properties that the behavior modeled in MECHATRONICUML needs to fulfill. Then, we perform two

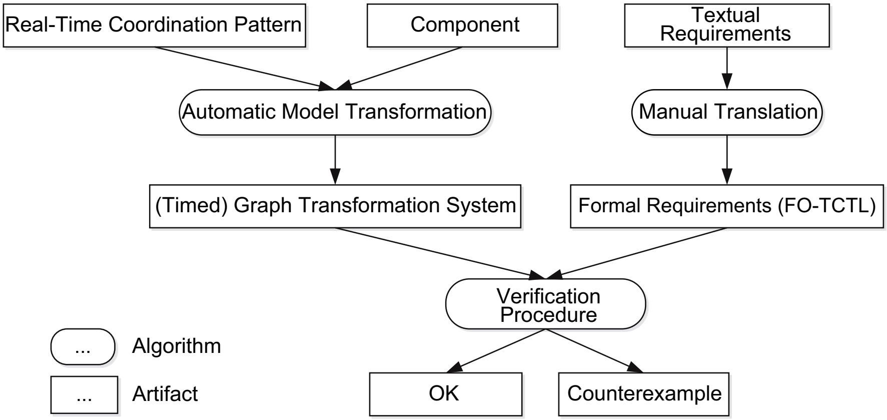

Real-Time Coordination Pattern
Component
Textual Requirements
Automatic Model Transformation
Manual Translation
(Timed) Graph Transformation System
Formal Requirements (FO-TCTL)
Verification Procedure
... Algorithm
... Artifact
OK
Counterexample

**Fig. 5.14** Overview of the design-time verification procedure

transformation steps that transform the inputs such that they can be processed by the verification procedures. The transformation of the real-time coordination pattern to a (timed) graph transformation is completely automatized. The transformation of textual requirements to formal requirements is a manual task. After explaining them in the following subsections, we will describe our verification procedure.

The result of applying this method to a MechatronicUML model is either that the model fulfills the formalized requirements or a counterexample. A counterexample is an execution of the system that leads to a state that violates the specified requirement. The counterexample is intended to support an engineer in locating and correcting the cause of an error in the model. After correcting the error, this method needs to be applied again until no more errors are found in the model. Then, the model is correct with respect to the formal requirements that have been verified. The verified model is the input for a code generator that generates the source code for the system.

### From Real-Time Statecharts to Graph Transformation Systems

Design-time verification of MechatronicUML models needs to capture the behavior of the real-time statecharts as well as their run-time reconfiguration operations in terms of component story diagrams. This is because both strongly influence each other. As an example, consider the real-time coordination pattern *ConvoyCoordination* shown in Fig. 5.9. The *coordinator* needs one subrole instance for each convoy member. Therefore, the multirole real-time statechart calls a reconfiguration operation as a side effect (cf. Fig. 5.12). The reconfiguration operation, in turn, creates a new subrole instance including an instance of the subrole real-time statechart. The execution of the new real-time statechart instance contributes to the behavior of the real-time coordination pattern instance and, thus, needs to be analyzed by the verification procedure. To cope with this strong interconnection between timed state-based behavior and reconfiguration, we use **timed graph transformation systems** (timed GTS), as shown in [53].

At this point, graph transformations [184] play a double role in our approach [53]. While they are used to model reconfiguration operations formally in terms of component story diagrams, they are also used as a meta-language to define the semantics of MECHATRONICUML. Such formally defined semantics is the basis for an automated verification procedure. A key extension necessary for self-optimizing mechatronic systems is the annotation of time, which is needed to capture the semantics of real-time statecharts. Therefore, we use timed GTS as the basis for our verification procedure. The use of timed GTS at this level, however, is hidden from the modeler who gives a MECHATRONICUML specification to the model checker which performs the translation automatically.

A timed GTS [53] consists of a start graph, a type graph, and a three different types of rules, namely timed graph transformation rules (timed GT rules), clock instance rules, and invariant rules. The start graph defines the starting point for the execution of the timed GTS and the type graph defines the types of nodes and edges for all graph generated by the timed GTS. Timed GT rules change a timed graph, but may neither add nor remove clock instances. Clock instance rules are used to add all clock instances that are possibly required for the application of a timed GT rule. Invariant rules forbid the existence of a subgraph of a timed graph after a certain time bound. For a formal definition of timed GTS, we refer to our technical report [196].

The translation of a MECHATRONICUML model into a timed GTS needs to encode the behavior of the real-time statecharts by timed graph transformation rules. We use objects representing the instances of the real-time statecharts including their states. Transitions cause a change of the active state of a real-time statechart. Consequently, we create a timed GT rule for each transition of a real-time statechart. State invariants forbid that a state is active beyond a specified point in time. They are translated to invariant rules. The clocks that are used by the real-time statecharts are created using clock instance rules. We refer to [53] and our technical report [92] for more information on the translation.

## From Textual Requirements to Formal Requirements

As shown in Fig. 5.14, a second translation is required for translating the textual requirements into formal requirements. Informal requirements in natural language are not suitable for being processed by an automatic verification procedure. An automatic verification procedure requires a formal specification of the requirements. Such translation needs to be carried out manually by an engineer. For timed automata, TCTL [6] has been introduced as a formal language for expressing such requirements.

In our *ConvoyCoordination* example, operating in convoy mode requires one RailCab to operate as a coordinator and periodically send reference data updates to all other convoy members. There, we need to ensure, e.g. that after the coordinator sends an update to a member, the coordinator must receive an acknowledgement within 50 ms. This constraint is formalized by the TCTL property

$$\mathbf{AG}(\text{coordinator.sendUpdate} \Rightarrow \mathbf{AF}_{<=50}\text{coordinator.idle})$$

shown in Fig. 5.9. This property obviously needs to be valid *for all* subrole instances of the *coordinator* and, in particular, must be valid throughout the reconfiguration.

In our verification procedure, we use FO-TCTL which is an extension of TCTL [6] by constructs of first-order logic. It enables specification of properties on graph structures in a much more user-friendly way compared to plain TCTL. In particular, it supports specifying a property that needs to be valid *for all* subrole instances of a real-time coordination pattern. To achieve this, we introduce variables that range over the nodes of a graph, constants that represent particular nodes that are known at design-time, predicates representing types of nodes and edges, and quantifiers. Variables allow the formulation of properties concerning nodes without knowing which particular nodes exist during run-time. Expressing the same property using the normal TCTL requires knowledge of all nodes that may exist during the execution of the system.

Verification Procedures

We defined two verification procedures for verifying properties specified in FO-TCTL based on a timed GTS that we will introduce in the following subsections. The first verification procedure, called FO-TCTL model checking, uses a state-exploration technique that enumerates the run-time states of the timed GTS, thereby considering the timing conditions of the timed GTS. Consequently, it supports timed GTS with a finite number of run-time states. Our second verification procedure applies a shape analysis technique. It supports timed GTS with an infinite number of run-time states, but it does not consider the timing conditions.

**Verification Procedure 1 – FO-TCTL Model Checking:** Our FO-TCTL model checking procedure consists of three steps that are visualized in Fig. 5.15. The key idea of our approach is a reduction of the model checking problem for a timed GTS and a FO-TCTL specification to the well-studied TCTL model checking problem for timed automata [6, 7, 21]. Then, a standard timed model checking tool answers the question whether the MECHATRONICUML model fulfills its formal requirements.

In the first step, a so-called Gt-automaton is computed for the timed GTS. The Gt-automaton is a timed automaton where each of its states corresponds to a timed graph which can be derived from the initial graph of the timed GTS. Transitions result from derivations using the timed GT rules and are labeled with the guard and reset of the timed GT rule that was used for the derivation. We label each state with the clock constraints of the invariant rules that can be matched to the state. Each node in a state is labeled with a unique identifier that is preserved by the derivation. The set of clocks of the Gt-automaton corresponds to the union of the clock instances that have been created by the clock instance rules.

In the second step, we use the Gt-automaton to reduce the FO-TCTL formula to a standard TCTL formula. In particular, we exploit the identifiers of the nodes for replacing quantifiers and variables by boolean expressions with constants, only. An $\exists$ quantifier is replaced by a disjunction replacing the occurrences of the quantified

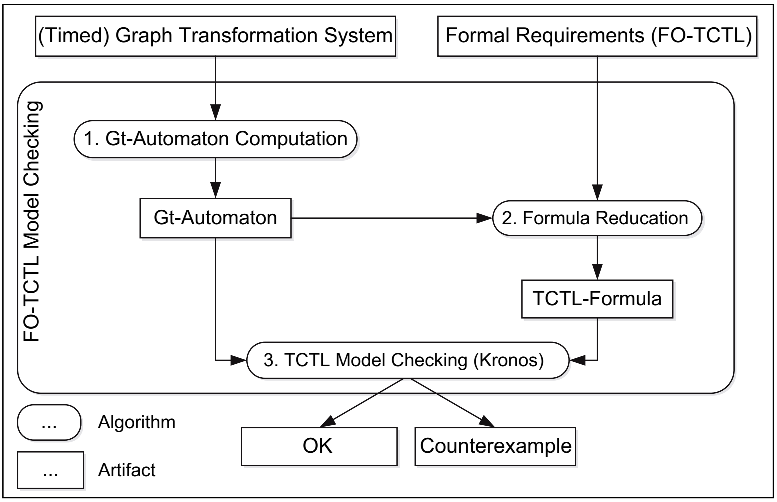

(Timed) Graph Transformation System
Formal Requirements (FO-TCTL)
FO-TCTL Model Checking
1. Gt-Automaton Computation
Gt-Automaton
2. Formula Reducation
TCTL-Formula
3. TCTL Model Checking (Kronos)
OK
Counterexample
...
Algorithm
...
Artifact

**Fig. 5.15** Overview of FO-TCTL model checking

variable by all possible node identifiers occurring in the Gt-automaton. A $\forall$ quantifier is replaced analogously by a conjunction. Finally, we encode the identifiers by atomic propositions that can be processed by the timed model checker.

In the third step, we use the Gt-automaton and the TCTL formula as inputs to a standard timed model checking tool. We propose using Kronos [31] because Kronos provides a full TCTL model checking. UPPAAL [40], on the contrary, only supports a simple subset of TCTL. UPPAAL can be used with our method as well, if the supported TCTL subset is sufficient for the verification task.

**Verification Procedure 2 – Shape Analysis:** Commonly, models contain behavior that allows the runtime structure specified by the model to grow. An example of this is convoy coordination in the RailCab system, where new RailCabs can join existing convoys. Usually, and in this example as well, there is no natural limit to this growth.

There are two ways out of this. One is to use bounded model checking, which is what the method detailed above amounts to. Instead of checking the entire system, a finite subsystem is identified by bounds, such as maximum convoy size, and then checked. In order to construct the Gt-automaton, the entire state space of the system must be constructed, and thus all possibilities of infinite growth pruned at some arbitrary bound, like, e.g. 10 RailCabs. Any behavior within that subsystem is safe, yet nothing is known of the remainder. That means that any correctness result obtained using this method is only valid as long as there is no convoy longer that 10 RailCabs. As soon as there is, all verification results obtained with this bound in place are lost.

In this particular case, there might be some merit to limiting the number of RailCabs that can take part in a convoy a priori to a constant number. This is because the communication range of RailCabs is limited, as is the maximum deceleration

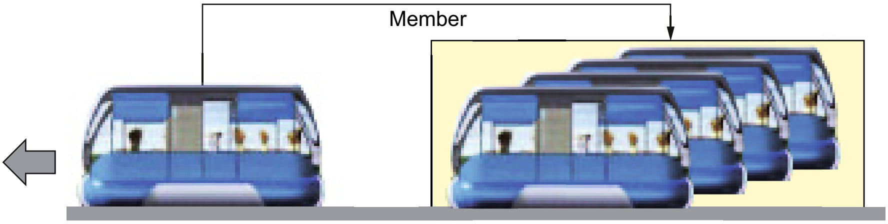


**Fig. 5.16** An abstracted convoy

a RailCab is capable of, which limits the length of a potential convoy. However, such limitations usually only apply to hardware structures, such as convoys. Also, as the system evolves, physical parameters change. Better transmitters might extend a RailCabs WiFi range, improved brakes might improve maximum deceleration. Most successful distributed systems eventually outgrow any bound on size.

The second way to deal with infinite growth is called overapproximation, and that is what *shape analysis* essentially does. Instead of looking at a subset of the behavior of a system, in shape analysis one looks at a superset of it which has the property of being compactly (finitely) representable. This is done in such a way that safety properties that can be shown for the overapproximation, are also guaranteed to be valid for the original system.

Shape Analysis was initially a formalism used to abstractly describe heap structures in imperative programs [185]. In our work we have utilized the concepts developed for that formalism to create a verification algorithm that applies them to GTS [193, 194, 211]. This algorithm is generic and applies to all GTS. It is an instance of a class of algorithms performing abstract graph transformations. Other instances include [25, 137, 176].

At its core, the algorithm works by identifying groups of nodes in a given graph that have similar properties and grouping them together into one *summary node*. The resulting graph then is a *representative* for the set of all graphs where the summary node is replaced by a particular number of nodes. Thus a single abstract graph, called *shape*, can represent an infinite number of actual, concrete graphs. As an example, consider Fig. 5.16. Here, the rectangle represents an arbitrary number of follower RailCabs. The entire shape therefore represents a convoy of arbitrary size.

Such shapes can now be subjected to dynamic behavior, just as the original graphs were. If the abstraction was chosen well, we obtain a finite representation of the entire state space and can check then whether the given safety properties are valid or not. If they are, we have just proven the safety of the original system in its unconstrained form, e.g. the safety of convoy coordination regardless of the number of participants. If they are not, we get a counterexample. This counterexample can either be genuine, or it can be an artifact of the abstraction. This can be decided by retracing the counterexample obtained on the shape level on concrete graphs. If the counterexample is genuine, we need to fix the system, if it is not, we must refine the abstraction to remove the artifact that produced the counterexample.

The ability to verify infinite systems does of course not come without a price. The two main drawbacks of this method are increased complexity and undecidability. Abstraction introduces a lot of complex definitions and properties that make it hard to enrich with additional properties. This is the reason Shape Analysis is currently unable to take time into account in any form (unlike the method described above). Undecidability means that in its finished form, the algorithm will run fully automatically, but in the absence of human intervention there is the possibility that the algorithm will run forever. It is however possible to reduce the probability of this by allowing the algorithm access to as much domain specific information as possible to help it guide its abstraction refinement process.

## *5.2.4 Discrete Behavior*

After proving the correctness, the real-time coordination patterns are integrated into the component implementation and refined if necessary. We provide an algorithm for checking the correctness of the refinement in Sect. 5.2.4.1. Additionally, we may integrate existing legacy components into a MechatronicUML model such that they meet the system's safety and liveness requirements; this is presented in Sect. 5.2.4.2. In Sect. 5.2.4.3, we will provide an automatic synthesis of component behavior to resolve dependencies that might exist between different real-time coordination patterns when they are combined in a component. Sect. 5.2.4.4 describes the specification of reconfiguration behavior of components. Finally, we will outline a planning technique that selects which runtime reconfigurations to apply to reach the system's objectives at runtime in Sect. 5.2.4.5.

### 5.2.4.1 Refinement of Real-Time Coordination Patterns

Real-time coordination patterns as introduced in Sect. 5.2.3 aim at reusing the modeled interaction in different applications. Therefore, real-time coordination patterns are specified independent of a concrete component implementation. A concrete implementation often has to refine this behavior, e.g. add internal computations or access internal variables, thereby introducing new internal states and/or transitions. Such modifications of the behavior may invalidate the formal requirements that have been proven for the real-time coordination pattern using the design-time verification procedure (cf. Sect. 5.2.3.2).

As described in Sect. 5.2.3.1, real-time coordination patterns may specify 1:*n* communication with runtime reconfiguration. Then, the reconfiguration operations need to be considered when checking for a correct refinement [91]. This problem is more difficult than the 1:1 communication case as additionally the creation and deletion of the protocols and the dependencies between the instances has to be considered. Since the abstract real-time coordination patterns are verified formally beforehand, the refinement must preserve these verified properties.

The overall refinement approach is shown in Fig. 5.17. First, a real-time coordination pattern is modeled as described in Sect. 5.2.3.1. Afterwards, we verify this real-time coordination pattern using the verification approach outlined in Sect. 5.2.3.2

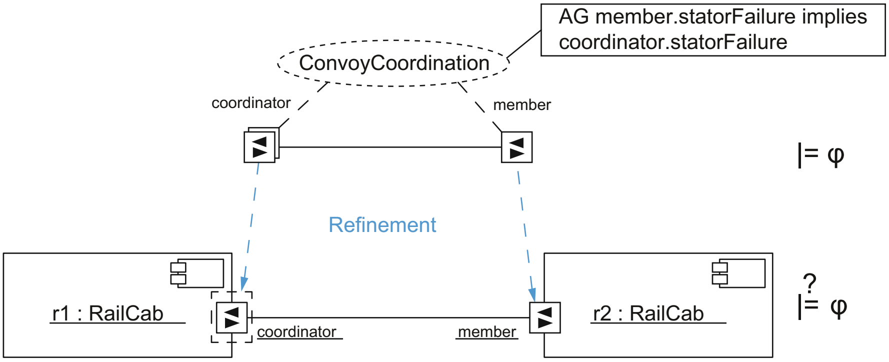


**Fig. 5.17** Refinement approach

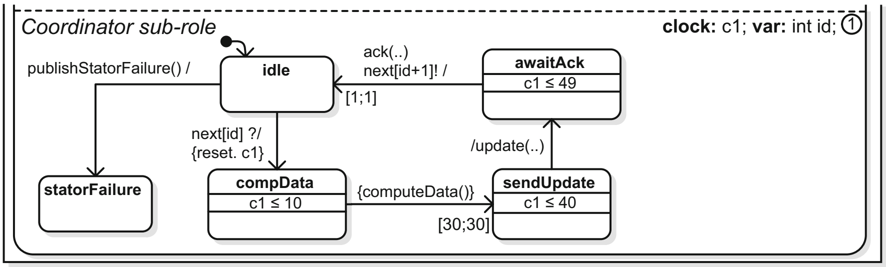


**Fig. 5.18** Excerpt of the refined real-time statechart of the convoy coordinator

for proving that the specified properties $\varphi$ are valid. Then, both roles are refined to a port as part of a component implementation. Finally, we check the conformance of the component implementation to the roles of the real-time coordination pattern by checking for a correct refinement.

As an example, consider the excerpt of a *coordinator* real-time statechart shown in Fig. 5.18. The real-time statechart has been refined with respect to the real-time statechart shown in Fig. 5.12 by inserting a new state *compData*. The transition from *compData* to *sendUpdate* specifies a side effect that computes new data to be sent via the *update* message which causes the transition to consume 30 time units of computation time. Since the timing values have been changed and a new state has been added, it is not clear whether the refined *coordinator* multirole still fulfills all verified properties.

In the literature, two basic types of refinements are defined: *simulation* and *bisimulation* that exist for untimed systems as well as for real-time systems [37, 202]. These standard refinement definitions are based on automata and disregard run-time reconfiguration that is used in our approach. Additionally, simulation is a very weak condition as it does not require the refined system to specify all communications being specified in the abstract real-time coordination pattern. Obviously, this is not sufficient for safe protocol reuse. In contrast, bisimulation is a very strong condition as it requires the refined system to perform exactly the same in exactly the same time as the abstract real-time coordination pattern. This does not allow applying changes to the protocol, thereby limiting the set of component implementations complying to the abstract real-time coordination pattern.

Therefore, we introduce a refinement definition called *relaxed weak timed bisimulation* [91] that relaxes the strict conditions of a bisimulation by using information of our component model. We assume that each port has an unbounded input buffer for received messages that can accept messages at any time. If a statechart receives a message, the message is taken out of the input buffer. Using such an input buffer, the point in time, at which a message is consumed by a real-time statechart, does not matter for a communication partner. Therefore, we allow that the refined role processes messages later than the abstract role. Delaying a sent message is not allowed as we only consider one role in the refinement and we cannot assume that the receiver of the message can still receive it after the time interval specified by the abstract real-time coordination pattern has elapsed.

We consider the run-time reconfiguration of real-time coordination patterns by a so-called structural refinement as defined in [90]. It ensures that the refined real-time coordination pattern executes its reconfiguration operations in the right time intervals by relating the subrole and singlerole instances including their connections in both the abstract and refined real-time coordination pattern instance.

Checking for a correct refinement requires checking the refined role implemented in a port of a component against the abstract role of the real-time coordination pattern. This requires exploring the state-spaces of both and to compare the intervals in which messages are sent or received. The refinement check algorithm is based on the same implementation as the verification procedure introduced in Sect. 5.2.3.2. In [91], we showed for the RailCab example that this is more efficient than verifying all properties for the refined real-time coordination pattern again. The reason is that we do not need to consider the connector, but only one role at a time.

#### 5.2.4.2 Integration of Legacy Components

The software of self-optimzing mechatronic systems is usually a network of components. By MECHATRONICUML we provide a sound method that guarantees a high quality of this software. However, in domains like the automotive industry the development of new functions is an exception rather than the norm. In many cases, components exist and have to be reused where no model or only incomplete models exist. On the one hand, reuse accelerates the development of the system. On the other hand, one can rely on the quality the component has proven in the past. Both saves development costs.

These so-called *legacy components* must be integrated into the newly built system such that they meet the system's safety and liveness requirements. Therefore, we reconstructed a real-time statechart that specifies the communication behavior of the legacy component. The reconstructed real-time statechart is used to verify the correct integration of the legacy component into a MECHATRONICUML model [96, 97].

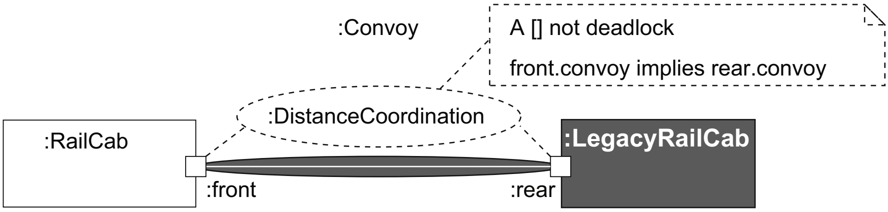


**Fig. 5.19** Architecture with legacy RailCab

Fig. 5.19 shows a scenario where an old RailCab *LegacyRailCab* communicates with a *RailCab* developed with MechtronicUML. Here we assume that the developer does not have a MechtronicUML model of the communication software of the *LegacyRailCab*. Both RailCabs shall communicate using the *DistanceCoordination* Pattern. The communication behavior of the rear role must satisfy the liveness constraint that no deadlock occurs (`A[] no deadlock`) and the safety contraint that both RailCabs drive in convoy mode (`front.convoy implies rear.convoy`) when applying the *DistanceCoordination* pattern.

The role behavior with which the legacy component has to interact is called *context*. In Fig. 5.19 the context is the front role of the component RailCab. An integration is successful, if the communication between the legacy component and the context is error-free. This is specified by safety properties and liveness properties. Moreover we need to guarantee that, depending on the communication behavior, the correct control behavior is executed. The continuous behavior is identified by system identification.

In order to integrate a legacy component into a MECHATRONICUML model, the following requirements must be met. The legacy component must provide an interface that is accessible by the developer. This interface must define all incoming and outgoing messages used for communication, all signals used by embedded feedback controllers, and all information which is relevant for executing the component (e.g. execution periods). This, however, does not require additional effort in the domain of safety-critical systems, as this is typically part of the system specification.

Moreover, we assume that initially the component is in its starting state or in a quiescent state (cf. [127, 213]). We further assume that the developer is able to put the component in such a state.

The information provided by the interface of the legacy component may differ substantially. We distinguish three cases. First, *(1)* the interface provides functionality to query its current state. If this is not the case, we distinguish the cases where *(2)* the source code of the interface is provided and *(3)* no source code is provided.

Depending on the provided information, we apply different methods to integrate the legacy component. For case *(1)* we apply grey-box-checking, for case *(2)* white-box-checking, and for case *(3)* black-box-checking. We will shortly explain these methods below. We will introduce the basic approach of iterative learning by grey-box-checking. Thereafter we will point out how the other methods differ from grey-box-checking.

### Grey-Box-Checking

We start grey-box-checking with a *chaotic closure*. This chaotic closure is an over-approximation of the actual communication behavior. The chaotic closure is a behavior model that enables all possible communication behaviors and also a deadlock of the legacy component at any time. However, not all of this behavior may be implemented in the legacy component. Therefore, the behavior is defined step-by-step by limiting the behavior of the chaotic closure until it conforms to the behavior of the legacy component.

First, we verify the safety and liveness properties on the combination of the chaotic closure and the context. If the verification yields a counterexample, the counterexample is used to generate a test case for the legacy component. The test case is generated by extracting all inputs and outputs including their time or appearance. The legacy component is executed with the extracted inputs. The test has passed, if the extracted outputs are observed from the legacy component at identical points of time as in the counterexample. Otherwise, the test has failed.

If the test case passed, we have found a valid counterexample. This means, one of the required safety and liveness properties are not satisfied. At this point, reverse engineering either stops or the requirements on the system need to be relaxed. If the test case failed, the observed behavior is used to refine the chaotic closure. Therefore, the current state is requested from the legacy component. If a new state is found, a new state is created for the chaotic closure. Edges are built according to the observed transitions. This process continues until a valid counterexample is found or all traces of the context have been taken into account.

### Black-Box-Checking

Black-box-checking also uses a counterexample guided refinement. But here the legacy component does not provide the functionality to request its current state. Our solution is to construct a candidate for the behavior of the legacy component. The candidate is constructed by an extension of the learning algorithm of Angluin (1987) [9], an efficient approach for learning a deterministic finite automaton of a black-box. We extended the algorithm of Angluin (1987) to take into account incoming and outgoing messages and time.

The candidate and the context are verified by model checking with respect to the safety and bounded liveness properties of the legacy component. If the verification is successful, it is proven that the candidate is equivalent to the behavior of the legacy component. Otherwise, the counterexample is used to improve the candidate.

### White-Box-Checking

For White-box-checking, we assume that we know the source code of the legacy component. To safely integrate the legacy component into the system, we generate source code from the context model. The source code of the context and the legacy component are embedded into a framework. The framework simulates scheduling, message exchange and timed behavior. The resulting system is verified by a source code model checker with respect to the safety and bounded liveness properties.

### 5.2.4.3 Synthesis of Component Behavior

As described in Sect. 5.2.4.1, the roles of a real-time coordination pattern are refined to ports of a component. Often, a component needs to engage in more than one interaction in order to fulfill its function, i.e. it refines roles of several real-time coordination patterns.

In the RailCab example, a RailCab interacts with other RailCabs for building convoys, but it also needs to register at the track section it is currently driving on. The registration at track sections is required to ensure that each track section is only accessed by RailCabs driving in the same direction. Up to this point, all interactions defined by real-time coordination patterns are operating independent of each other. For a safe convoy operation, however, we need to fulfill the requirement that "in convoy operation mode, each participating RailCab has to be registered to a track section" [54]. Thus, there may exist dependencies between real-time coordination patterns when they are combined in a component.

In our previous works, we used a so-called **synchronization real-time statechart** in a component for resolving such dependencies [75]. The specification of such a synchronization real-time statechart was subject to the developer. Specifying a synchronization real-time statechart, however, is a difficult and error-prone task. This is because, on the one hand, it needs to resolve the dependency and, on the other hand, it must not remove communications specified by one of the roles. If communications specified by one of the roles was removed, the results of the design-time verification will not necessarily be valid anymore.

As a solution, we provide an automatic synthesis of a component behavior that automatically resolves the dependencies and ensures the *role conformance* of the resulting behavior [54]. The dependencies are either specified by *state-composition rules* referring to the states of the roles or by *event-composition automata* referring to the sent and received messages of the roles. To perform the synthesis, first, a product automaton including the behavior of all ports is constructed and the dependencies are resolved automatically by applying the state-composition and event-composition rules. Then, the role conformance check ensures that all communications originally specified by both roles are still available in the synthesized behavior. That, in turn, ensures that all verified properties are still valid in the synthesized behavior.

We illustrate our approach by a simplified example using UPPAAL timed automata [21]. Fig. 5.20a) shows a simplified convoy behavior consisting of the two states *noConvoy* and *convoy*. Fig. 5.20b) shows a timed automaton for registering at a track section.

In the following, we will first introduce state-composition rules. Thereafter, we will explain event-composition automata before outlining the synthesis algorithm.

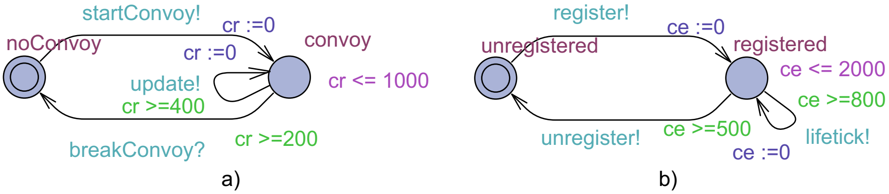


**Fig. 5.20** Simplified behavior models for a) convoy coordination and b) registration

### State-Composition Rules

State-composition rules define restrictions for the component behavior based on the states of the role automata. In particular, they specify forbidden state combinations of the input automata. The state information encoded in state-composition rules includes timing information that forbids certain state combinations only for a specific time interval. The state-composition rules need to be specified by a developer when creating the component.

An example of a state-composition rule for the automata in Fig. 5.20 is given by:

$$r_1 = \neg((unregistered, true) \wedge (convoy, true)).$$

The state-composition rule $r_1$ formalizes the requirement that a RailCab may only drive in convoy mode while it is registered at a track section. Consequently, $r_1$ forbids the component behavior to be in states *unregistered* and *convoy* at the same time. A reference to a state, e.g. $(unregistered, true)$, is a tuple where the first entry refers to the name of the state and the second entry defines a clock restriction. In this case, the clock restriction is *true* for both states which means that the state combination is not allowed for all possible clock values of all used clocks.

### Event-Composition Automata

Event-composition automata define restrictions for the component behavior based on sequences of messages that the role automata may send or receive. They monitor the messages that are sent or received by the role automata. Consequently, only messages used in one of the role automata may be used in an event-composition automaton.

In our example, we assume another requirement for the RailCab component which states that a RailCab needs to be registered at a track section for at least 2500 time units before it can start at convoy. This requirement refers to a *message* of the automaton and, thus, can not be specified by state-composition rules. The event-composition automaton for the requirement is shown in Fig. 5.21.

The automaton starts in state *ec_initial*. If it monitors that the RailCabs sends *register*, it switches to *ec_registered* thereby setting the clock *ec_c1* back to 0. The message *startConvoy* may be monitored, at the earliest, 2500 time units later which is specified by the time guard of the corresponding transition. The automaton stays in *ec_registeredConvoy* until the RailCab sends an *unregister* message.

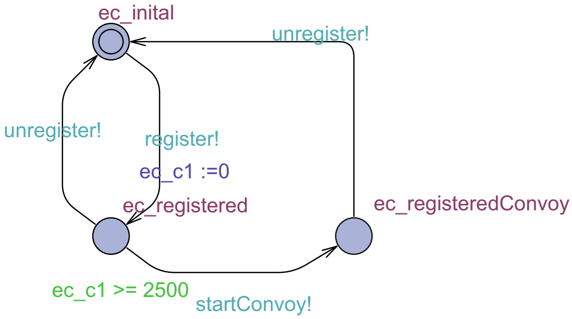


**Fig. 5.21** Example of an event composition automaton $eca_1$

Synthesis Algorithm

The synthesis algorithm takes three inputs. These are the automata of the roles that should be synthesized to a component behavior as well as the state-composition rules and the event-composition automata that define the restrictions for the synthesis. Based on these inputs, the resulting automaton for the component is synthesized in four steps. We briefly outline these steps in the following. For a complete description, we refer to Eckardt and Henkler (2010) [54].

**Step 1 – Computing the parallel composition:** In the first step, we compute the parallel composition of the role automata. The parallel composition is derived from the parallel composition operator of CCS (Calculus of Communicating Systems, [144]) which is also used in UPPAAL [21]. The parallel composition contains the complete behavior of the role automata.

**Step 2 – Applying state-composition rules:** In the second step, the state-composition rules are applied to the parallel composition resulting from step 1. Iteratively, each state-composition rule is applied to each state of the parallel composition automaton. If the state fulfills the state conditions imposes by the state-composition rule, the time condition is added to the invariant of the state. If the resulting invariant of the state is false, the state is removed from the parallel composition along with all its incident transitions.

The state $(convoy, unregistered)$ of the product automaton fulfills the state conditions of $r_1$. Consequently, the time conditions $\neg((true) \wedge (true))$ is added to the invariant which makes it $false$ and causes the state to be removed.

**Step 3 – Applying event-composition automata:** In the step 3, all event-composition automata are applied iteratively to the automaton resulting from step 2. Since the event-composition automaton is a timed automaton, the application is similar to the parallel composition of step 1. The difference is that the event-composition automaton only monitors the messages that are sent and received by the role automata.

After the parallel composition, each state refers both, to the states of the initial parallel composition and the state of the event-composition automaton. All states resulting from the parallel composition that are not reachable from the initial location are removed from the automaton.

**Step 4 – Checking Behavior Conformance:** In step 4, we check for behavior conformance of the synthesized automaton resulting from steps 1-3. In step 2 and step 3, every behavior not allowed by the state-composition and event-composition rules have been removed from the parallel composition of the role automata. Due to the removal of behavior, it is not ensured that the communications specified by the roles of the real-time coordination pattern are still contained in the synthesized automaton. As a consequence, it is not guaranteed that the synthesized behavior still fulfills all properties that have been verified for the real-time coordination pattern.

By checking the synthesized automaton for behavior conformance to the roles of the real-time coordination pattern, we ensure that the verified properties still be valid. *"In order to preserve the relevant role behavior, we need to ensure that in the refined component behavior, every timed safety properties and every untimed liveness properties are preserved. This would imply that no deadlines of the original role automata are violated while all events of the original automata are (in the correct order) still visible within the original time interval. If both of these properties are preserved, we say that the refined component behavior is role conform."* [54]

If the synthesized automaton resulting from steps 1-3 is not behavior conform, the synthesis reports an error. In this case, it is not possible to synthesize an automaton that fulfills the state-composition and event-composition rules while specifying the behavior of the roles of the real-time coordination pattern. In this case, the engineer needs to specify the synchronization real-time statechart manually and ensure correctness by repeating the verification steps introduced above.

We refer to our technical report [55] for a detailed proof of the correctness of the synthesis.

### 5.2.4.4 Modeling Component Reconfiguration

In Sect. 5.2.3 we showed how reconfiguration is specified for real-time coordination patterns. Additionally, we can specify reconfiguration behavior for the components of our component model [43]. For the specification of reconfiguration behavior, we use component story diagrams [200] again. We specify component story diagrams for each component of the component model that needs to perform reconfiguration.

Figure 5.22 shows an example of a component story diagram that specifies the behavior for becoming a convoy member for the *DriveControl* component in Fig. 5.8. Becoming a convoy member requires to instantiate the subcomponent *MemberControl* and to connect it to the *SpeedControl* such that it can provide the reference speed for the speed controller.

Since we model the component reconfiguration by component story diagrams, we can use our design-time verification procedure introduced in Sect. 5.2.3.2 for verifying the reconfiguration behavior. We refer to [94] for more technical information on executing reconfiguration in a hierarchical component model.

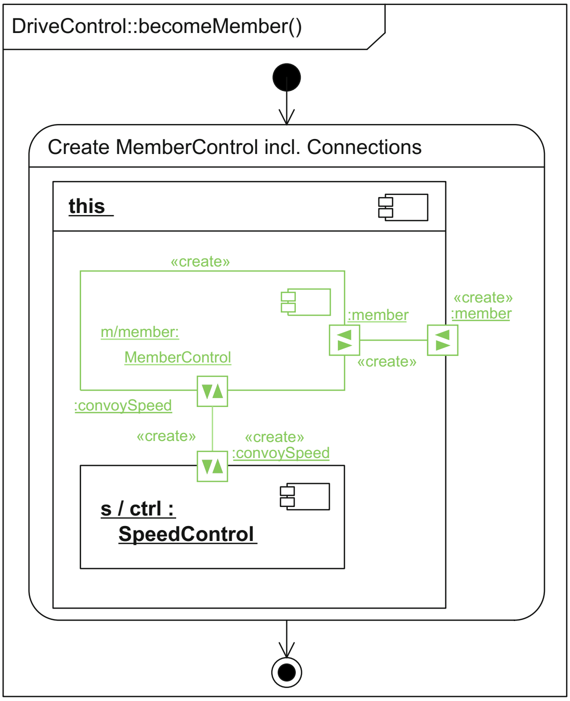


**Fig. 5.22** Reconfiguration rule of *DriveControl* to become a convoy member

### 5.2.4.5 Safe Planning

In each configuration of a self-optimizing mechatronic system, a large set of runtime reconfigurations can be applied to adapt the system to changes in its environment at runtime. Selecting which runtime reconfigurations to apply can be a complex task. Self-optimizing systems often have superordinated objectives that should be reached during execution, like optimizing the energy consumption or achieving user-specified objectives. These objectives have to be respected when selecting which runtime reconfigurations to apply. However, selecting runtime reconfigurations that are likely to help to achieve the objective is no trivial task. Since the selection of runtime reconfigurations is supposed to happen autonomously (a human intervention would not meet the response time requirements of self-optimizing mechatronic systems), it has to be planned by a software system.

To prevent unsafe configurations, e.g. an inadequate safety distance between two RailCabs, from occurring in a plan, the planning system should further take safety requirements into account. The safety requirements restrict the set of valid configurations, i.e. they specify which configurations are not allowed to occur in a resulting plan. In contrast to the verification of runtime reconfigurations (cf. Sect. 5.2.3.2), where the absence of unsafe states is guaranteed categorically, this technique allows unsafe states to exist in the reachability graph, but plans the reconfigurations in such a way that no unsafe state is passed through. The latter approach is chosen for specific safety requirements that can not be verified by the design-time verification or impose to many restrictions to the specification of the runtime reconfigurations. In [69, D.o.S.O.M.S. Sect. 3.2.9] , we present a technique that considers these safety requirements when planning runtime reconfigurations.

Our approach uses GTS as an underlying formalism. The transition system of the GTS can be constructed by successively applying the graph transformations to the initial configuration and its successor configurations. The planning task is to find a path in this transition system so that a target configuration is reached. A safe planning task is basically the same, but includes the requirement that no potentially unsafe configuration is passed through. In our case, the initial configuration corresponds to a UML object diagram and each transition is the result of applying a graph transformation rule to the configuration.

To solve these planning tasks, different algorithms and techniques exist. One of the approaches is to translate the planning problem into an available off-the-shelf planning system. These traditional planning systems, however, employ models different from GTS. They employ models with first-order literals that are usually compiled into a propositional representation by grounding predicates and actions. While a translation is basically possible, there are some restrictions because typical planning languages, like the Planning Domain Definition Language (PDDL), which is the current *de facto* standard in academia, has a different expressive power than GTS. By planning directly in the transition system defined by the GTS, we avoid these problems.

Given a goal specification, our model can be fed into a planning system, e.g. [57], that directly plans on the transition system that results from the model. Therefore, no translation to a dedicated planning language and thus no restriction to the expressive power of GTS is necessary. Unsafe configurations are recognized by the planning system and not allowed in a valid plan. The resulting plan specifies a sequence of runtime reconfigurations that safely turn the system from its initial configuration into a target configuration.

### 5.2.5 Simulation of Hybrid Behavior

The software of a self-optimizing mechatronic system consists of discrete software developed with MechatronicUML and continuous controllers developed with a tool like MATLAB/Simulink. That results in a so-called hybrid behavior specification [98]. The design-time verification procedures described in Sect. 5.2.3.2 can only be applied to the discrete software. Corresponding hybrid verification techniques [5] do not scale sufficiently for complex mechatronic systems.

Therefore, we use simulation for testing the complete hybrid system and, in particular, the correct integration of discrete software and controllers. We provide an automatic model transformation for transforming the MechatronicUML model into an input of a simulation tool, namely MATLAB/Simulink [93, 95].

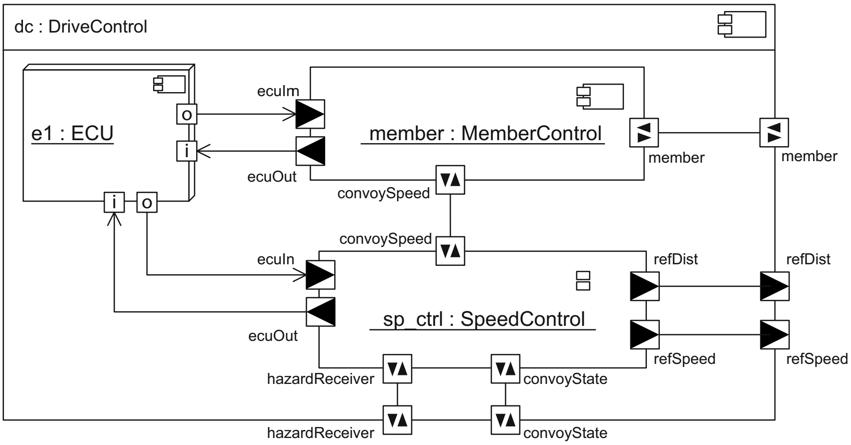


**Fig. 5.23** Deployment diagram

### 5.2.6 Specification of Deployment

The software components that have been created with MECHATRONICUML need to be deployed on a hardware. This hardware comprises sensors that provide input signals, controllers that control actuators and computing hardware that executes the software components. In MECHATRONICUML the deployment of software to hardware is specified by deployment diagrams. Hardware entities are represented by hardware nodes which communicate unidirectionally via hardware ports.

Figure 5.23 shows an example of a deployment diagram which specifies the deployment of an instance of the component type *DriveControl* (cf. Fig. 5.8) to an ECU. In deployment diagrams hardware nodes are drawn as boxes. Hardware ports are drawn as squares that contain either an "i" for incoming signals or an "o" for outgoing signals. The embedded component instances *member:Member* and *sp_ctrl:SpeedControl* of *dc:DriveControl* are both connected to the hardware node *e1:ECU* which represents the ECU that executes these component instances.

### 5.2.7 Integration of Self-healing Behavior

The self-optimization capabilities of self-optimizing mechatronic systems can be used to repair systems in case of failures at runtime. This so-called self-healing can be used to reduce occurrence probabilities of hazards in systems which are applied in safety-critical environments. Self-healing systems react to failures by a reconfiguration of the system architecture during runtime.

Take for example the speed control of the RailCab. The electric current to be set on the linear drive depends on the speed of each wheel which is measured by speed sensors. If a failure occurs in at least one of the speed sensors, a wrong value is passed to the current controller. This causes the RailCab to drive at a wrong speed which can result in a collision. To prevent such a situation, a self-healing operation can be specified in form of a reconfiguration which replaces the faulty sensor by a spare which is still working.

This reaction is subject to hard real-time constraints because reacting too late does not yield the intended self-healing effects. Consequently, it is necessary to analyze the propagation times of failures and the effect of a reconfiguration on the propagation of failures [166]. In [69, D.o.S.O.M.S. Sect. 3.2.13] , we present an approach for the analysis of self-healing operations which specifically considers these properties.

Not all parameters which are needed to analyze self-healing operations, e.g. the concrete system architecture, are known at design time. When, for example, RailCabs have become ready for the market, they will be produced by more than one manufacturer. Then it will be possible that two vehicles that come from different manufacturers meet on the track. In order to build a convoy they need to establish a connection. This connection leads to a system architecture that was unknown at design time, because the system architecture of the unknown vehicle was, of course, unknown to the developers of the RailCab.

Consequently, the effect of self-healing operations needs to be analyzed during runtime. We developed an approach to analyze self-healing operations at runtime. It prevents the construction of system architectures at runtime where self-healing operations can not reduce the occurrence probabilities of hazard so that they become acceptable.

Based on the system's current architecture, we compute each reachable system architecture for a fixed number of subsequent reconfigurations at runtime. We then analyze the self-healing operations. If the hazard occurrence probability of a reachable system architecture exceeds the system's acceptable hazard occurrence probability event after the application of a self-healing operation, the reconfiguration rule that constructs this system architecture is locked.

### *5.2.8 Code Generation*

We use the models that have been created using MECHATRONICUML for an automatic generation of the source code of the self-optimizing mechatronic system. An approach for code generation has been introduced in [1]. Alternatively, we can use the MATLAB/Simulink code generation facilities to generate code out of the MATLAB/Simulink model that we created for simulation (cf. Sect. 5.2.5).

## 5.3 System Optimization

Harald Anacker, Michael Dellnitz, Kathrin Flaßkamp, Philip Hartmann, Christian Horenkamp, Bernd Kleinjohann, Lisa Kleinjohann, Martin Krüger, Sina Ober-Blöbaum, Christoph Rasche, Maik Ringkamp, Robert Timmermann, Ansgar Trächtler, and Katrin Witting

In order to develop self-optimizing systems, optimization plays a crucial role. In the following section, a couple of methods are presented which allow a systematical and formal optimization of the system behavior. In contrast to successive improvement, which often has to be done manually, these methods aim at automatically seeking the optima, i.e. points of no further improvement. During the conceptual design, the relevant objectives are identified and a general control structure is designed, that is capable to alter the fulfillment of the objectives, cf. Sect. 3.2. At the beginning of the system's design and the development, concrete mathematical models of the system behavior are created in the respective domains as well. These are the inputs for the methods of **model-based self-optimization** described in the first seven sections. The following sections deal with behavior-oriented self-optimization which describes methods without an explicit physical model of the system or process. Instead, these approaches work on mapping input values to output values. The actual system and the considered process are observed as a black box.

In the first section, we get back to multiobjective optimization which has already been introduced in Sect. 1.4.1.1 and present some more details about novel set-oriented algorithms for solving multiobjective optimization problems (MOP) in Sect. 5.3.1. The algorithms can be used to compute optimal system configurations that considers several conflicting objectives in one single MOP.

Self-optimizing systems are often complex systems consisting of several subsystems which are hierarchically structured (see Sect. 1.3 for an introduction into the structuring concept). If each system comes with its own objectives, one also gets a hierarchy of MOPs that has to be solved. Section 5.3.2 describes an approach on how to handle such optimization problems. The following section, Sect. 5.3.3, is closely related to hierarchical optimization. A so-called hierarchical model is introduced that can be used to significantly reduce the model complexity of hierarchical systems by means of parametric model-order reduction.

The following two Sections 5.3.4 and 5.3.5, deal with MOPs which also depend on continuous external parameters. Two numerical methods are presented that are used to solve such problems efficiently and to identify so-called robust Pareto points.

Optimal Control, a different aspect of system optimization, is addressed in Sections 5.3.6 and 5.3.7. In optimal control problems the goal is to compute time-dependent steering maneuvers as introduced in Sect. 1.4.1.2. In Sect. 5.3.6, the optimal control technique DMOC is presented which is especially tailored for the optimal control of mechanical systems. In order to improve the solvability of optimal control problems by creating efficient initial guesses, a motion planning technique based on motion primitives is described in Sect. 5.3.7. Within this approach, several short pieces of simply controlled trajectories are sequenced to longer trajectories.

In Sect. 5.3.8 one approach for decision making (cf. Sect. 1.4.1.3) is presented that is called hierarchical hybrid planning. The hierarchical model is used to simulate the prospective system behavior and a discrete planning problem is defined based on the simulation results as well as on a pre-computed Pareto set.

All these different methods of system optimization need a physically motivated mathematical model of the self-optimizing system. If such a detailed model is not available for a specific task, methods of the **behavior-oriented self-optimization** can be used (see Sect. 1.4.2 for an introduction). Statistical Planning, described in Sect. 5.3.9, is one of these methods. It uses statistical data to compute plans for mechatronic systems based on an environmental model given by a discrete finite nondeterministic Markov decision process. A different approach is presented in Sect. 5.3.10. A discrete planning problem is defined that can be used to find a sequence of operation modes which describe a transition from an initial state to a predetermind goal state. Section 5.3.11 presents an approach to realize a multiagent system by behavior planing, to open up the advantage given by the possibility for intelligent communication of individual subsystems. Finally we will present the application of solution pattern presented in Sect. 4.5 to make the method hybrid planning available to the developers.

## *5.3.1 Set-Oriented Multiobjective Optimization*

Michael Dellnitz, Kathrin Flaßkamp, and Christian Horenkamp

The demand for multiobjective optimization in the context of self-optimizing systems was already shown in Sect. 1.4.1. Here, we review algorithms developed and applied within the CRC 614 for the computation of the entire Pareto set of multiobjective optimization problems. The basic idea of these methods is to use set-oriented algorithms for dynamical systems (cf. [46]).

We reconsider the multiobjective optimization problem (MOP Eq. (1.1)) introduced in Sect. 1.4.1

$$\min_{\mathbf{p}\in S\subset\mathbb{R}^n} \mathbf{F}(\mathbf{p}), \tag{5.1}$$

where $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^k$, $\mathbf{F}(\mathbf{p}) = (f_1(\mathbf{p}),...,f_k(\mathbf{p}))^T$ is the vector of $k \in \mathbb{N}$ objective functions, $\mathbf{p}$ the optimization variable or design variable of dimension $n \in \mathbb{N}$, and $S$ denotes the feasible set. The necessary conditions for Pareto optimality are given by the Karush-Kuhn-Tucker (KKT) equations (cf. Sect. 1.4.1). Here, we consider the left hand side of the KKT equations as a map $\mathbf{H} : \mathbb{R}^{k+n} \to \mathbb{R}^n$, with $\mathbf{H}(\beta,\mathbf{p}) = \sum_{i=1}^{k} \beta_i \nabla f_i(\mathbf{p})$ and $\beta = (\beta_1,...,\beta_k)$ with $\beta_i \geq 0$ for all $i \in \{1,...,k\}$ and $\sum_{i=1}^{k} \beta_i = 1$ (cf. Eq. (1.2) in Sect. 1.4.1). By finding zeros of the map $H$, we identify points that satisfy the necessary optimality conditions. Therefore, the use of zero finding strategies as well as the minimization of the function $H$ are essential steps in many techniques for solving multiobjective optimization problems.

### 5.3.1.1 Set-Oriented Solution Techniques for Multiobjective Optimization

The set-oriented solution techniques for multiobjective optimization are implemented in the software package GAIO[11]. They can be divided into two approaches: subdivision methods and recovering methods, which we shortly introduce in the following (cf. [190] for a detailed overview).

[11] Global Analysis of Invariant Objects, see `www.math.upb.de/~agdellnitz`

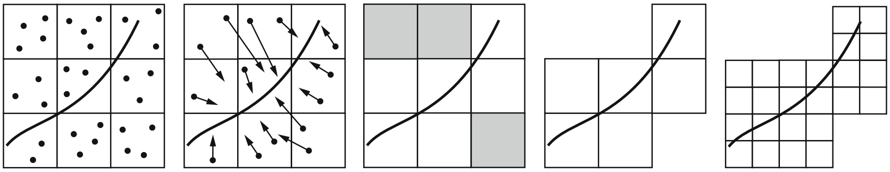


**Fig. 5.24** Illustration of the subdivision algorithm: it alternates between subdivision and selection steps to approximate the Pareto set by a box covering

The **subdivision** procedure (cf. Fig. 5.24 for a sketch) starts with a box that covers the admissible set of optimization parameters and approximates the Pareto set by a successive refinement and selection of boxes. After every subdivision step, a gradient method is applied to chosen test points in all boxes. This iterates the test points forward, possibly into other boxes. The selection step deletes all boxes that do not contain iterated test points and only keep the other boxes for further subdivision. This scheme generates a box covering of the Pareto set with desired refinement. A sampling algorithm (cf. [190]) that does not require gradient information can be used instead of the gradient step.

**Recovering** techniques are applied to fill gaps in the covering of the Pareto set. Under certain conditions, the Pareto set (locally) forms a manifold [100], i.e. in the neighborhood of already known Pareto points further points can be found. The recovering algorithm is similar to a predictor corrector method, which is typically used for numerical integration. Based on an initial partial box covering, new test points are generated nearby (prediction step) and then iterated until they fulfill the KKT conditions (correction step). In this way, connected components of the Pareto set can be found if at least one Pareto point of this component is already known (cf. Fig. 5.25).

In the following two sections, we present two basic strategies to use the recovering technique. Firstly, the recovering techniques are applied in the preimage space (space of optimization parameters). This approach reaches its limitations when the number of design variables is high. In such a case one can pursue a second strategy, for which the recovering techniques are applied in the image space (space of objective functions). This is more suitable if the number of design variables is high but the number of objectives is small as discussed in Sect. 5.3.1.3.

### 5.3.1.2 Set-Oriented Recovering Methods in the Preimage Space Applied to the Multiobjective Optimization of the Test Vehicle Chameleon

In this subsection we review a recovering technique for the approximation of the Pareto set based on a predictor corrector method working in the preimage space. This method is a very efficient tool for the approximation of a finite representation of the entire Pareto set and has been successfully applied for the multiobjective optimization of the test vehicle Chameleon within the CRC 614.

In principle, starting with a point $\mathbf{p}^*$ of the Pareto set, the following steps are performed:

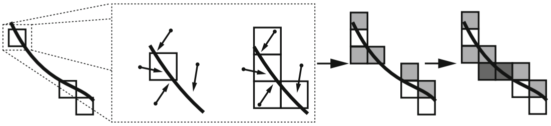


**Fig. 5.25** Illustration of the recovering algorithm: starting with an initial partial covering of the Pareto set, a full covering is computed by a generation and mapping of test points near existing boxes

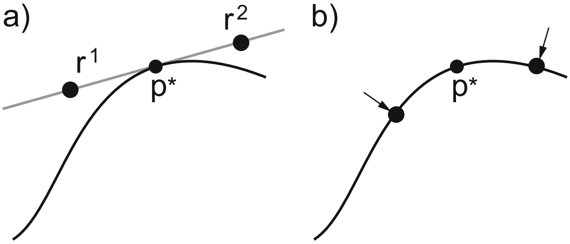


**Fig. 5.26** Illustration of the predictor-corrector method: (a) Predict points $\mathbf{r}^1$ and $\mathbf{r}^2$ in the neighborhood of $\mathbf{p}^*$. (b) Correct points such that they lie on the Pareto set.

1. Predict points $\mathbf{r}^1,...,\mathbf{r}^m$ in the neighborhood of $\mathbf{p}^*$.
2. Correct the points $\mathbf{r}^1,...,\mathbf{r}^m$ such that they lie on the Pareto set by minimizing the norm of the KKT equations and adding the boxes containing the corrected points.

An illustration of this technique can be found in Fig. 5.26. The number of predicted points $m \in \mathbb{N}$ has to be sufficiently large in order to cover the Pareto set after the correction step well. Therefore, the bottleneck of this method is the prediction step where new test points are generated near an initial solution $\mathbf{p}^*$. A common way for the generation of new test points is to linearize the Pareto set around the initial solution $\mathbf{p}^*$ by an approximation of the tangent space of the Pareto set in $\mathbf{p}^*$ (*grey line in Fig. 5.26*). In general, one can use the Hessians of all objectives, but this approach reaches its limitation when high-dimensional models, where $n$ is large, are considered. In the course of the research of the CRC 614, a novel method has been developed for the treatment of high-dimensional MOPs by successive approximation of the tangent space [181]. In detail, a new algorithm has been stated, where the tangent space is approximated by secants. This algorithm leads to an efficient approximation of the Pareto set of high-dimensional MOPs. Table 5.1 shows the CPU time for a scalable multiobjective optimization problem with three objectives which are taken from [189] for the recovering algorithm using the tangent space approximation ($R_C$) and the new algorithm ($R_S$) developed in [181]. For the test problem a significant speedup can be obtained for large $n$.

Within the corrector step in which the predicted points are corrected such that they lie on the Pareto set, many efficient minimizers make use of derivatives of the objective function $\mathbf{F}$. In many applications only program code for the objective $\mathbf{F}$ is provided and the corresponding derivatives, if existent, can not be determined analytically, thus other techniques are required. For example, finite differences can

**Table 5.1** Comparison between the classical recovering algorithms $R_C$ and a method using a successive approximation of the tangent space $R_S$

| dimension of MOP | | $R_C$ | $R_S$ |
|---|---|---|---|
| 100 | CPU time | 2.9 | 2.9 |
| 200 | CPU time | 14 | 11.9 |
| 500 | CPU time | 134 | 91 |
| 1000 | CPU time | 965 | 500 |

be used, however, this approach leads to inaccurate derivatives which slows down the correction step. Alternatively, algorithmic differentiation (also called automatic differentiation) can be used (cf. [81]). These techniques automatically compute formulas for the derivatives based on the program code of the optimization problem for example.

In [182] the recovering technique of [181] has been combined with algorithmic differentiation. In more detail, the feasible set $S$ of a MOP has been described as a zero set and the recovering procedure is adapted as follows: Let $\mathbf{p}^*$ be a solution of the MOP (1.1), then for the prediction step select neighboring points of $\mathbf{p}^*$ along the feasible set $S$ and correct them to points on $S$. After the correction step a non-dominance test is performed to ensure that only the Pareto set is approximated as a subset of the feasible set $S$. For all non-dominated points, the predictor-corrector step is repeated until a covering of the Pareto set is reached. For the correction of the predicted points, the derivatives involved are calculated by an algorithmic differentiation method.

This method was successfully applied for the multiobjective optimization of the distribution of the tire forces for a braking maneuver of the test vehicle, Chameleon, which is described in more detail in Sect. 2.3. The tire forces of the Chameleon can be influenced individually within the physical and technical restrictions [175], hence there are a multitude of possibilities to realize a braking maneuver with the same braking force. In [182] the slip $\lambda_i$ and the slip angle $\alpha_i$ for each wheel $i = 1,...,4$ are the optimization parameters. The objectives are to avoid tire wear by minimizing the squared sum of the slip angles ($f_1$) and for each tire the minimization of the distance between the tire force and the adhesion limit for safety reasons ($f_2,...,f_5$). Fig. 5.27 shows projections of the resulting Pareto set and Pareto front.

Another extension of both the recovering and subdivision algorithms is the use of parallelization techniques. This is motivated by time-consuming function evaluations of a sufficiently high amount of test points involved in the algorithms. In [26], for instance, a multiobjective optimization problem is solved for the resource efficient design of integrated circuits. In more detail, the dimensions of transistors in simple logic cells are optimized with respect to noise margin, propagation delay and dynamic energy consumption. A function evaluation in this setup is a

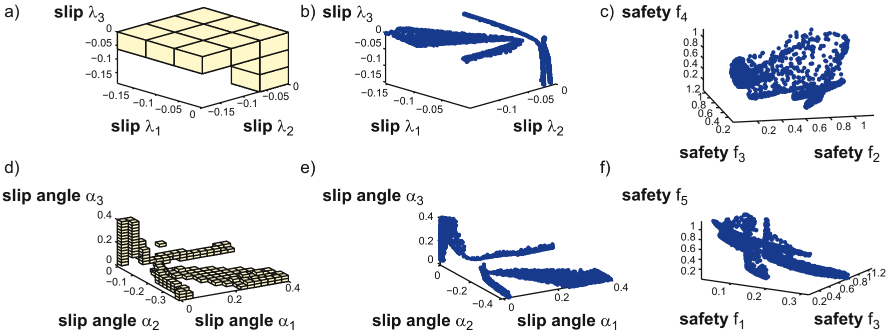


**Fig. 5.27** Projection of the resulting Pareto set and Pareto front of a multiobjective optimization problem of the distribution of the tire forces for a braking maneuver of the test vehicle Chameleon. The Pareto set and Pareto front were computed with the algorithmic differentiation approach: (a), (d) A set of boxes covering smooth connected parts of the Pareto set. (b), (e) Corresponding Pareto points. (c), (f) Corresponding Pareto front. Figure from [182].

one to three seconds simulation of an integrated circuit. Using a parallelization infrastructure, it is possible to obtain good approximations of the Pareto set within adequate computational time.

#### 5.3.1.3 Set-Oriented Recovering Methods in the Image Space Applied to the Multiobjective Optimization of the Active Guidance System of the RailCab

The previously described recovering method reaches its limitations if the number of design variables is high. In such a case, the approximation of the tangent space in the predictor step is computationally costly, therefore the recovering method will be applied in the image space Image space (cf. [41]). The principal procedure is the same as shown in Fig. 5.26. This approach is a good alternative for the case when the dimension of the parameter set is high and only a few objectives are considered.

This method was applied to find trajectories of the RailCab vehicle (cf. Sect. 2.1) in the rails [72, 206]. The control of the RailCab vehicle is done by the active guidance system that controls the displacement of the vehicle in the rails. It controls the position of the front and rear axles. The computed trajectories should maximize safety ($f_1$) and passenger comfort ($f_2$) and minimize the average energy consumption ($f_3$) of the hydraulic actuators. Naturally, this problem is an optimal control problem but due to the fact that the problem underlies a certain structure it can be transformed into a multiobjective optimization problem with a high number of parameters (cf. Sect. 1.4.1.2 and 5.3.6). In [72] the trajectories of the front and rear axles of a fixed rail track with respect to $f_1, f_2$ and $f_3$ have been optimized. Due to the high amount of parameters, a recovering method in the image space Image space is necessary. In Fig. 5.28 the computed Pareto front is shown. Two optimal compromise solutions were selected (marked by a circle and a rectangle). In Fig. 5.29 the

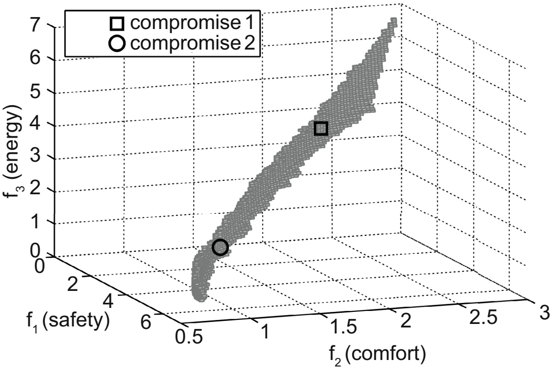


**Fig. 5.28** Pareto front for the three objectives safety, comfort and energy. The trajectories corresponding to the Pareto points marked with a rectangle and a circle are shown in Fig. 5.29. Figure from [72].

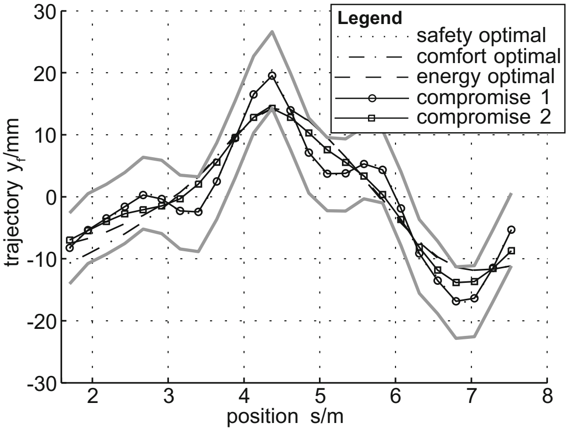


**Fig. 5.29** Different reference trajectories for the front axle. While comfort prioritizing trajectories are apparently smooth, safety prioritizing trajectories try to follow vertical displacement to stay near the middle line. Figure from [72].

corresponding trajectories of the position of the front axle are shown. The recovering techniques in the image space are also suitable to find well-distributed Pareto points in the image space. In [183] such a method was applied to design an operating strategy for the Energy Management of a Hybrid Energy Storage System combining batteries and double layer capacitors.

To sum up, various applications have shown the great suitability of the set-oriented mutliobjective optimization methods in the design of self-optimizing technical systems.

## *5.3.2 Hierarchical Multiobjective Optimization*

Michael Dellnitz and Maik Ringkamp

Modeling of self-optimizing systems often leads to hierarchical multiobjective optimization problems. These kinds of problems consist of several MOPs instead of just one MOP. All MOPs are related to each other by a hierarchy. The solutions of a lower level MOP restrict the preimage space of the next higher level MOP in the sense that the feasible set of the higher level MOP is a subset of the lower level Pareto set. Each level of the hierarchy consists of one MOP.

Consequently, in the case of two MOPs two levels of hierarchy exist. Such a problem is also called bilevel MOP and is defined as follows:

$$\begin{aligned} \min_{(\mathbf{p},\mathbf{p}^1)\in\mathbb{R}^n\times\mathbb{R}^{n_1}} \quad & \mathbf{F}(\mathbf{p},\mathbf{p}^1) \\ s.t. \quad & (\mathbf{p},\mathbf{p}^1)\in S \\ & \mathbf{p}^1 \in \mathscr{P}_{\mathbf{f}^1}(\mathbf{p}) \end{aligned} \tag{5.2}$$

Here, the **p**-dependent Pareto set $\mathscr{P}_{\mathbf{f}^1}(\mathbf{p})$ is defined as the solution of the MOP of the lower level:

$$\mathscr{P}_{\mathbf{f}^1}(\mathbf{p}) := \arg\min_{\mathbf{p}^1\in\mathbb{R}^{n_1}} \quad \mathbf{f}^1(\mathbf{p},\mathbf{p}^1) \tag{5.3}$$

$$s.t. \qquad (\mathbf{p},\mathbf{p}^1)\in S^1 \tag{5.4}$$

with feasible sets $S,S^1 \in \mathbb{R}^n\times\mathbb{R}^{n_1}$, objective functions $\mathbf{F}:\mathbb{R}^n\times\mathbb{R}^{n_1}\to\mathbb{R}^k$, and $\mathbf{f}^1:\mathbb{R}^n\times\mathbb{R}^{n_1}\to\mathbb{R}^{k_1}$.

Under given regularity conditions, bilevel MOPs can be solved using the Karush-Kuhn-Tucker equations (Eqs. (1.2)) of the lower level MOP as additional equality constraints for the upper level MOP as described in detail in [42].

The hierarchical structure of the optimization problems derived from the OCM structure allows to consider a special case of the general bilevel MOP (5.2). Instead of computing one general MOP on the lower level, we consider problems where the lower level MOP can be separated into several independent MOPs, i.e. each MOP has a different set of optimization parameters.

#### 5.3.2.1 Hierarchical Multiobjective Optimization by Parametrization of the Lower Levels

More specifically, the kind of problems we consider are given as

$$\begin{aligned} \min_{(\mathbf{p},\mathbf{p}^1,\dots,\mathbf{p}^l)\in S} \quad & \mathbf{F}(\mathbf{p},\mathbf{p}^1,\dots,\mathbf{p}^l) \\ s.t.\ & \mathbf{p}^j \in \mathscr{P}_{\mathbf{f}^j}, j\in\{1,\dots,l\}, \end{aligned} \tag{5.5}$$

where $l \geq 1$ is the number of independent lower level MOPs and $S \subseteq \mathbb{R}^n\times\mathbb{R}^{n_1}\times\ldots\times\mathbb{R}^{n_l}$ the feasible set as in Eq. (5.2) with independent Pareto sets $\mathscr{P}_{\mathbf{f}^j}, j\in\{1,\dots,l\}$, as solutions of the $l$ lower level MOPs

$$\mathscr{P}_{\mathbf{f}^j} := \arg\min_{\mathbf{p}^j\in\mathbb{R}^{n_j}} \mathbf{f}^j(\mathbf{p}^j) \tag{5.6}$$

with objective functions $\mathbf{F} : \mathbb{R}^n \times \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_l} \to \mathbb{R}^k$ and $\mathbf{f}^j : \mathbb{R}^{n_j} \to \mathbb{R}^2, \forall j \in \{1, \ldots, l\}$.

Under certain regularity conditions, the resulting Pareto sets $\mathscr{P}_{\mathbf{f}^j}$ of the lower level MOPs are 1-dimensional submanifolds of $\mathbb{R}^{n_j}$ for each $j \in \{1, \ldots, l\}$. Thus, these sets can be parametrized by variables $\alpha^j \in [0, \alpha_{max}]$ and a map $\varphi^j : [0, \alpha_{max}] \to \mathscr{P}_{\mathbf{f}^j}$. The parametrization reduces the complexity of the upper level MOP, it can be described with the help of an auxiliary objective $\tilde{\mathbf{F}} : \mathbb{R} \times [0, \alpha_{max}] \times \ldots \times [0, \alpha_{max}] \to \mathbb{R}^k, \tilde{\mathbf{F}}(\mathbf{p}, \alpha^1, \ldots, \alpha^l) := \mathbf{F}(\mathbf{p}, \varphi^1(\alpha^1), \ldots, \varphi^l(\alpha^l))$ as

$$\begin{aligned} \min_{(\mathbf{p},\alpha^1,\ldots,\alpha^l) \in \mathbb{R}^n \times [0,\alpha_{max}] \times \ldots \times [0,\alpha_{max}]} \quad & \tilde{\mathbf{F}}(\mathbf{p}, \alpha^1, \ldots, \alpha^l) \\ s.t. \qquad (\mathbf{p}, \alpha^1, \ldots, \alpha^l) \in S. & \end{aligned} \tag{5.7}$$

For problem (5.5) we propose the following solution strategy:

1. Compute the Pareto sets of all independent lower level MOPs (5.6) by using the methods explained in Sect. 5.3.1.
2. Parametrize the resulting Pareto sets by the map $\varphi$.
3. Use the parametrization variables as parameters for the MOP on the next higher level and solve the auxiliary problem (5.7).

This method was successfully applied for example in [131] or [102] to solve bilevel MOPs derived by the OCM structure. In the latter work, the considered application examples are an active suspension system and a linear drive with an active air gap adjustment which both represent a module of the rail-bound vehicle RailCab. Hierarchical optimization is used to combine the module-related optimal operating strategies. In Fig. 5.30 the computed Pareto front of the upper level MOP is shown.

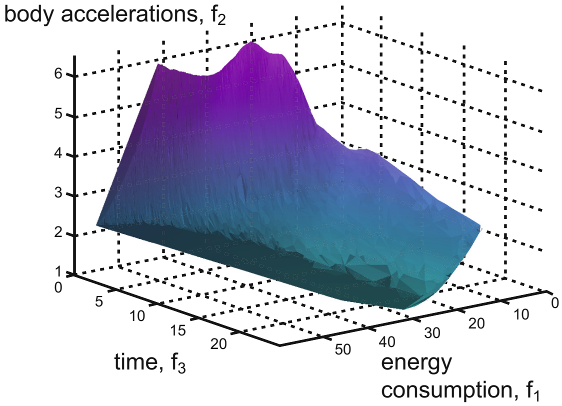


**Fig. 5.30** *Active suspension system and linear drive:* Computed Pareto front for the hierarchical model of the combination of the active suspension system and the linear drive with an active air gap adjustment (original figure from [102]).

## *5.3.3 Hierarchical Modeling of Mechatronic Systems*

Martin Krüger and Ansgar Trächtler

The hierarchical modeling is based on the hierarchical OCM structure presented in Sec. 1.3. Mathematical models of the dynamical behavior are needed for several methods in the design process of self-optimizing systems. Such methods are for example, the design of feed-forward or feedback controllers, identification and observation of system parameters respective states or model-based optimization. Complexity of the models rapidly increases at higher levels of the system hierarchy. The modeling approach described in the following sections yields a so-called hierarchical model which uses the hierarchical structure to reduce the model complexity in a systematic way. Particularly, in combination with hierarchical multiobjective optimization (cf. 5.3.2) a novel approach for parametric model-order reduction can be used.

### 5.3.3.1 Hierarchical Model

Each element of the system hierarchy is equipped with its own information processing described by an OCM. In general, this reduces the complexity of the information processing, as several tasks can be encapsulated. However, the dynamical behavior of a subsystem depends on the underlying elements (subsystems) in the hierarchy. Hence, the behavior of the underlying subsystems has to be taken into account in the modeling process.

The idea of the hierarchical model is to include the dynamics of the underlying systems in a simplified form, rather than considering all details. This reduces the complexity of the resulting model while ensuring that models have an appropriate amount of detail that can be used by model-based methods. Figure 5.31 illustrates the general idea.

Additionally, if each element of the hierarchy is seen as a self-optimizing system with its own objectives, a Pareto set, i.e. a set of optimal compromises, can be computed by applying multiobjective optimization. This Pareto set can then be used as additional information for the simplification of the system before it is transferred to the superordinated element. The parametric model-order reduction approach described below has been developed especially for this task. The result is a simplified respective reduced model which can, for example, be simulated much faster than the original model while maintaining a certain variability in view of the objectives.

### 5.3.3.2 Parametric Model-Order Reduction

In the following we will give a short overview about a particular parametric model-order reduction approach which yields parametric reduced models for the Pareto-optimal systems that was first published in [129]. The Pareto-optimal systems are those that correspond to the Pareto-optimal parameters $p^\star \in \mathscr{P}_F$. The general goal is to construct an approximation of these Pareto-optimal systems in terms of the parameterization variable $\alpha$, limited to the case of two objective functions. This

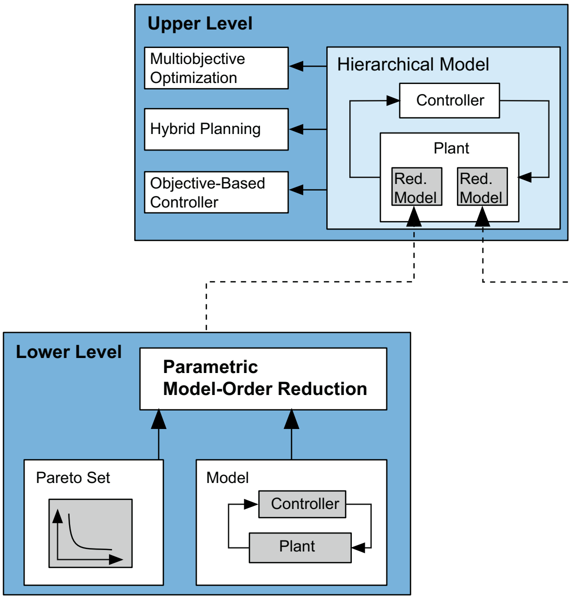


**Fig. 5.31** Hierarchical modeling principle for self-optimizing systems. First, the optimal configurations of the lower level module are computed. The resulting Pareto set is parameterized and the system model corresponding to the optimal configurations is reduced by parametric model-order reduction. On the upper level the reduced models are integrated in the hierarchical model which can then be used for following tasks as e.g. (hierarchical) optimization (Sect. 5.3.2) hybrid planning (Sect. 5.3.8) or the design of an objective-based controller (Sect. 2.1.4).

kind of model-order reduction can also be beneficial for analyzing the objective-based controller described in Sect. 2.1.4 where $\alpha$ is the control variable.

### Interpolation of Pareto-Optimal Systems

The parameterization of the Pareto set described in Sect. 5.3.2 also defines a parameterization of the Pareto-optimal systems. Assuming a linear closed-loop system

$$\dot{\mathbf{x}} = A(\varphi(\alpha))\mathbf{x} + B\mathbf{u}, \tag{5.8a}$$

$$\mathbf{y} = C\mathbf{x}, \tag{5.8b}$$

with $\mathbf{u}$ being the vector of external inputs and $\mathbf{y}$ being the output vector for calculating the objectives, the dynamics depend on the parameterization function $\varphi$. Since a higher number of parameters complicates the reduction process for almost all parametric model-order reduction algorithms we do not directly use this kind of parameterization. Instead, we create an interpolation, of the Pareto-optimal systems and not of the Pareto set, that depends directly on $\alpha$.

The first step is to define a sequence of knots $0 = \alpha_1 < \alpha_2 < \ldots < \alpha_k = \alpha_{max}$. Then, a component-wise linear spline interpolation can be applied to the Pareto-optimal systems that yields

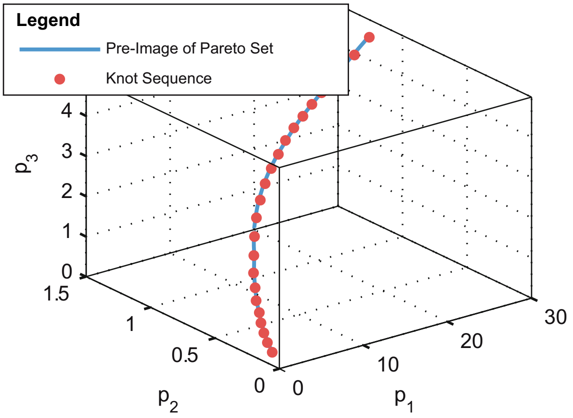


**Fig. 5.32** Pareto set of the active suspension system and results of the knot placement. Optimization parameters are given by three variables, which define the sky-hook damping of the system. A number of ten equidistantly placed knots has been used as input for the algorithm leading to a knot sequence of 24 knots placed along the Pareto set to reach the given error bound.

$$A(\alpha) := \underbrace{A(\varphi(\alpha_i))}_{A_i} + \frac{\alpha - \alpha_i}{\alpha_{i+1} - \alpha_i}\left[A(\varphi(\alpha_{i+1})) - A(\varphi(\alpha_i))\right] \tag{5.9}$$

for $\alpha \in [\alpha_i, \alpha_{i+1})$.

The number of knots $\alpha_i$ as well as their positions can be chosen automatically by an algorithm that is described in more detail in [129]. It consists of two parts. One part improves the knot positions of an existing sequence by means of the classical FORTRAN algorithm *newnot* [30] that has been extended to the matrix case. The second part compares the linear matrix-valued spline with a cubic one to estimate the approximation quality and inserts additional knots if necessary. Both parts are executed alternately until a given error bound is reached. Figure 5.32 shows the results of the knot placement for a Pareto set of the active suspension system, introduced in Sect. 2.1.4 using the same objectives energy consumption and level of comfort.

### Parametric Model-Order Reduction

The result of the aforementioned interpolation is a piecewise matrix polynomial $A(\alpha)$ and a corresponding parametric system

$$\dot{\mathbf{x}} = A(\alpha)\mathbf{x} + B\mathbf{u}, \tag{5.10a}$$

$$\mathbf{y} = C\mathbf{x}, \tag{5.10b}$$

with the states $\mathbf{x} \in \mathbb{R}^{n_x}$ and system matrices $A(\alpha), B$ and $C$ of appropriate dimensions that can be reduced by parametric model-order reduction. The first step of the reduction procedure comprises of a non-parametric reduction of the systems corresponding to the knots $\alpha_i$. Any projection-based reduction method that yields two projection matrices $V_i, W_i \in \mathbb{R}^{n_x \times q}$, can be used for this task, e.g. the IRKA (Iterative Rational Krylov Algorithm) to get an $\mathscr{H}_2$-optimal interpolation [10]. This leads to the reduced systems of order $q$

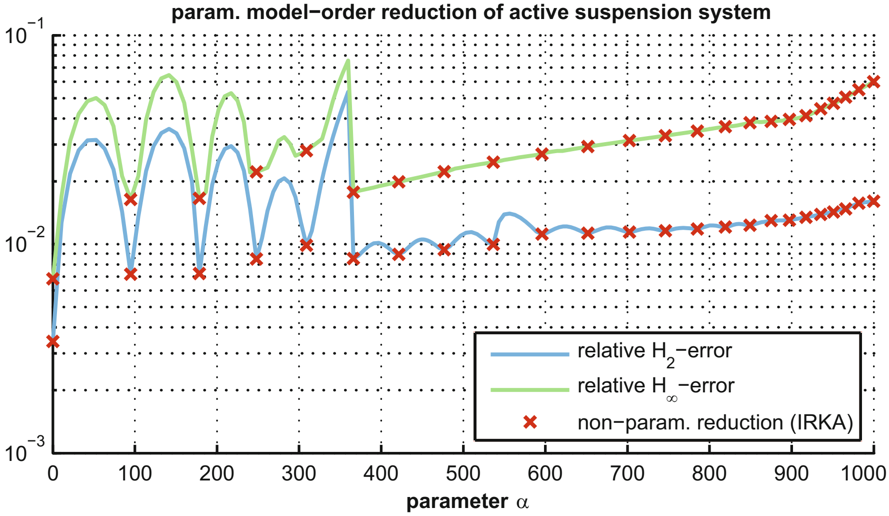


**Fig. 5.33** Relative error of the reduced system compared to the original system (5.10).

$$\underbrace{W_i^T V_i}_{\tilde{E}_{r,i}} \dot{\mathbf{x}}_r = \underbrace{W_i^T A_i V_i}_{\tilde{A}_{r,i}} \mathbf{x}_r + \underbrace{W_i^T B}_{\tilde{B}_{r,i}} \mathbf{u}, \tag{5.11a}$$

$$\mathbf{y} = \underbrace{C V_i}_{\tilde{C}_{r,i}} \mathbf{x}_r \quad (1 \leq i \leq k). \tag{5.11b}$$

Secondly, we apply a method called matrix interpolation to compute a parametric reduced system, see [160] for more details. Using Matrix Interpolation, the reduced matrices are compatible to one another by means of a reprojection to a common subspace, given by the columns of an orthonormal matrix $R \in \mathbb{R}^{n_x \times q}$. This matrix is computed by means of a singular value decomposition of the concatenation of the projection matrices $[V_1, \ldots, V_k]$. Each reduced system is then transformed by means of two quadratic matrices

$$M_i = (W_i^T R)^{-1} \; and \; T_i = R^T V_i. \tag{5.12}$$

The parametric reduced system consists of an interpolation of the transformed reduced matrices

$$E_{r,i} = M_i \tilde{E}_{r,i} T_i, \; A_{r,i} = M_i \tilde{A}_{r,i} T_i, \; B_{r,i} = M_i \tilde{B}_{r,i}, \; C_{r,i} = \tilde{C}_{r,i} T_i \tag{5.13}$$

In our case we use a simple weighted sum depending on $\alpha$, i.e.

$$A_r(\alpha) = \left(1 - \frac{\alpha - \alpha_i}{\alpha_{i+1} - \alpha_i}\right) A_{r,i} + \frac{\alpha - \alpha_i}{\alpha_{i+1} - \alpha_i} A_{r,i+1} \tag{5.14}$$

for the system matrix to give one example. The results of the parametric model-order reduction of the active suspension system are shown in Fig. 5.33.

### 5.3.4 Parametric Multiobjective Optimization

Michael Dellnitz, Christian Horenkamp, and Katrin Witting

Many mechatronic systems are subject to external forces or time-varying parameters. In many cases, such dependencies cannot be directly modeled in the optimization problem (5.1) in Sect. 5.3.1. Therefore, in this section, we extend the optimization problem (5.1) in such a way that it additionally depends on an external parameter $\lambda \in [\lambda_{start}, \lambda_{end}]$:

$$\min_{\mathbf{p}\in S} \mathbf{F}(\mathbf{p}, \lambda), \tag{5.15}$$

where $\mathbf{F}: \mathbb{R}^n \times [\lambda_{start}, \lambda_{end}] \to \mathbb{R}^k$, $\mathbf{F}(\mathbf{p}, \lambda) = (f_1(\mathbf{p}, \lambda), ..., f_k(\mathbf{p}, \lambda))^T$ is the vector of objective functions. The parameter $\lambda$ can model the dependence on time or any other external parameter of the objectives. In such situations, instead of choosing a single Pareto point, the decision maker has to choose a whole curve $\mathbf{p}(\lambda)$ describing for each $\lambda$ a Pareto optimal solution for the MOP. Similar as in Sect. 5.3.1, for each fixed $\lambda \in [\lambda_{start}, \lambda_{end}]$ the necessary optimality conditions are given by the Karush-Kuhn-Tucker equations, and the underlying optimization problem can be solved separately for each parameter value.

Consider the following parameter dependent MOP with $\lambda \in [0, 1]$ and the two objective functions $f_1, f_2 : \mathbb{R}^2 \times [0, 1] \to \mathbb{R}$ defined as

$$f_1(\mathbf{p}, \lambda) = \lambda\left((p_1 - 2)^2 + (p_2 - 2)^2\right) + (1 - \lambda)\left((p_1 + 2)^4 + (p_2 - 2)^8\right) \text{ and}$$
$$f_2(\mathbf{p}, \lambda) = (p_1 + 2\lambda)^2 + (p_2 + 2\lambda)^2.$$

Fig. 5.34 (a) shows the Pareto sets for different values of $\lambda$ and Fig. 5.34 (b) shows the entire $\lambda$-dependent Pareto set.

Calculating for each parameter value the entire Pareto set is numerically very costly and therefore, this approach is not suitable for applications, for which the solution has to be computed online. Thus, we propose a solution method which alternates between Pareto set computations and numerical path following of single Pareto points and therefore prevent the computation of the entire Pareto set. The proposed algorithm is designed for online use and works as follows:

1. Compute the entire Pareto set for a fixed parameter value $\lambda_1$ and select a point $\mathbf{p}(\lambda_1)$ on the Pareto set.
2. Compute the solution curve $\mathbf{p} : [\lambda_1, \lambda_2] \to S$ up to a fixed parameter value $\lambda_2$.
3. Compute the entire Pareto set for the parameter value $\lambda_2$ and select a point $x(\lambda_2)$ on the Pareto set. Proceed with step 2.

For the computation of the solution curve, in step 2 a predictor corrector method along the curve direction is involved.

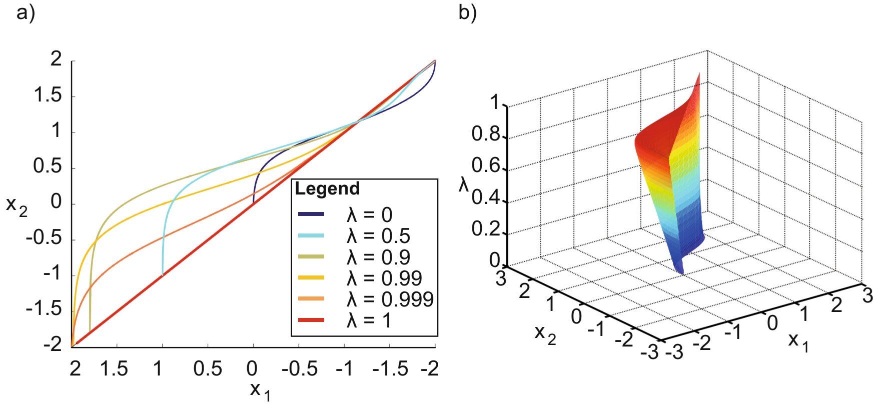


**Fig. 5.34** (a) Pareto sets for some specified values of $\lambda$. (b) entire $\lambda$ dependent Pareto set. Figure from [206].

The parameter dependent approximation of the Pareto optimal solutions was developed in [206]. In [208] and [187], it has been successfully applied to the optimization of the operating point assignment of the linear-motor of the driven railway system RailCab (cf. 2.1). It was also successfully applied to the active suspension system of the RailCab. In this application the crosswind has an influence and it was modeled as a parameter (see also Sec. 5.3.5.1).

### *5.3.5 Computation of Robust Pareto Points*

Michael Dellnitz, Robert Timmermann, and Katrin Witting

One important question in the context of multiobjective optimization problems (cf. Sect. 1.4.1.1) is the choice of the actual optimal configuration for one specific application, the so-called **decision making**. In this section we address this problem by defining the **robust Pareto points** and give a brief overview of two methods for the computation of such points. For a more detailed explanation, the reader is referred to [206]. We consider a Pareto point to be robust, if it varies as little as possible under variation of the external parameters of the parametric multiobjective optimization problem Eq. (5.15). Here, we additionally have the choice to regard the variation in parameter space or objective space.

Computation is based on two approaches: The first approach to the computation of robust Pareto points is based on numerical path following methods (cf. Sect. 5.3.4). First, a $\lambda$-dependent Pareto set for $\lambda = \lambda_{start}$ is computed. Secondly, $\lambda$ is varied from $\lambda_{start}$ to $\lambda_{end}$ for a subset of points of the Pareto set and the lengths of the resulting paths, which then run from the $\lambda_{start}$-Pareto set to the $\lambda_{end}$-Pareto set, are calculated. Finally, these path lengths can be taken into account when choosing one of the Pareto optimal operating points, since robust Pareto points are those with minimal path length. This enables the decision maker to choose points, which vary as little as possible under the influence of $\lambda$. If $\lambda$, for example, describes the influence of a change of temperature, one can chose an operating point, such that varying temperature has little effect on the system.

The second approach is based on the calculus of variations. The problem of finding the shortest path from a point on the Pareto set for $\lambda_{start}$ onto the Pareto set for $\lambda_{end}$ can be formulated as the variational problem

$$\begin{aligned} &\min_{(\mathbf{p}(\lambda),\alpha(\lambda))} \int_{\lambda_{start}}^{\lambda_{end}} \|\mathbf{p}'(\lambda)\|_2^2 \, d\lambda \\ &s.t.\, \mathbf{H}_{KT}(\mathbf{p}(\lambda),\alpha(\lambda),\lambda) = 0 \end{aligned} \tag{5.16}$$

where the constraint $\mathbf{H}_{KT} = 0$ represents the necessary Kuhn-Tucker equations for optimality (cf. Eq. (1.2)) in Sect. 1.4.1.1 and Sect. 5.3.1). The integral means, that the energy of the $\lambda$-dependent curve of Kuhn-Tucker points is minimized. If points exist in which all Pareto sets intersect, both approaches lead to the same robust Pareto points. Otherwise those points may differ. The main advantage of the second concept over the first one is that the starting point on the Pareto set needs not to be fixed in advance but is implicitly calculated during the minimization. Unfortunately, this concept is computationally more expensive, so if the underlying models are very complex or if execution time is critical (e.g. if the robust points are calculated in real time), the first concept is more suitable.

A much more detailed explanation of the path following approach can be found in [47], and two applications are presented in [26] (transistor sizing of CMOS logic standard cells) and [201] (robust Pareto points for the Active Suspension Module). For further reading about the variational method we refer to [207]. Both methods are also presented in [69, D.o.S.O.M.S. Sect. 3.1.8] .

### 5.3.5.1 Application

The second concept has been successfully used to compute robust Pareto points for the Active Suspension Module (ASM, cf. Sect. 2.1.4) in [130].

In this work, an external parameter $\lambda$ is used to model varying crosswind conditions which affect the ASM's behavior. A parametric multiobjective optimization problem was formulated using a simple ASM model with three degrees of freedom $p_1, p_2, p_3$ and with the two objectives comfort and energy consumption.

Figure 5.35 shows three Pareto sets for three different crosswind values and two robust Pareto points which were computed using the variational method. The robust point at $(0,0,0)$ corresponds to the energy optimal solution and could be expected in advance, the second point is nontrivial though and was not expected before the calculations. It can be used when designing the system such that it exhibits similar behavior in a variety of crosswind situations.

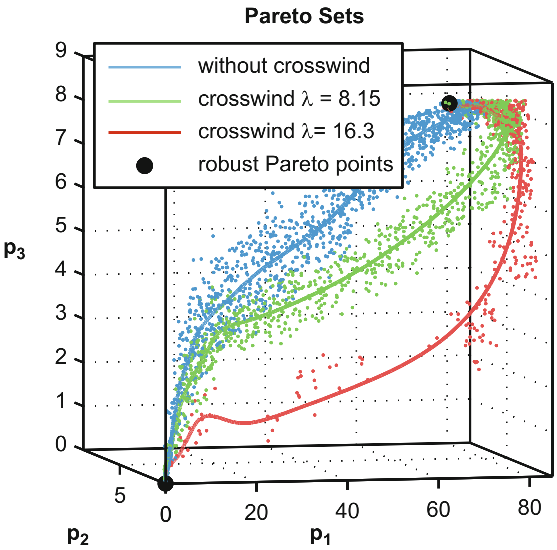


**Fig. 5.35** Application of the second concept (based on the calculus of variations) to compute robust Pareto points for the Active Suspension Module. This figure shows Pareto sets for three specific crosswind values and two robust Pareto points. Figure from [130].

### *5.3.6 Optimal Control of Mechanical and Mechatronic Systems*

Kathrin Flaßkamp and Sina Ober-Blöbaum

As introduced in Sect. 1.4.1, an optimal control problem seeks a control trajectory which steers the dynamical system in an optimal way with respect to a given cost functional. This is a challenging task for complicated nonlinear dynamical systems and thus has to be addressed by numerical techniques. In this section, we present an optimal control technique which is especially developed for the optimal control of mechanical systems (including mechatronic systems with additional electronic subsystems). For this class of systems, the equations of motion in the optimal control problem (OCP), cf. Eq. (1.3b), can be specified to the forced Euler-Lagrange equations, i.e.

$$\min_{\mathbf{x}(t),\mathbf{u}(t)} J(\mathbf{x},\mathbf{u}) = \int_0^T C(\mathbf{x}(t),\mathbf{u}(t))\,dt \tag{5.17a}$$

$$\text{with respect to } \frac{\partial L}{\partial \mathbf{q}}(\mathbf{q},\dot{\mathbf{q}}) - \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{f}(\mathbf{q},\dot{\mathbf{q}},\mathbf{u}) = 0 \tag{5.17b}$$

$$\mathbf{r}(\mathbf{x}(0),\mathbf{x}(T)) = 0, \text{ and} \tag{5.17c}$$

$$\mathbf{h}(\mathbf{x}(t),\mathbf{u}(t)) \leq 0 \text{ with } \mathbf{x} = (\mathbf{q},\dot{\mathbf{q}}). \tag{5.17d}$$

Here, the system's state $\mathbf{x} = (\mathbf{q},\dot{\mathbf{q}})$ consists of configurations $\mathbf{q}$ and corresponding velocities $\dot{\mathbf{q}}$, $L(\mathbf{q},\dot{\mathbf{q}})$ is the Lagrangian of the system (closely related to the system's energy) and $\mathbf{f}$ a control dependent forcing[12]. All possible configurations of a system form the configuration manifold[13] $Q$ such that the system's state space is given by the tangent bundle $TQ$.

To numerically solve an OCP, direct optimal control methods directly discretize the differential equations (5.17b). This can be done by integration schemes, i.e. the continuous state $\mathbf{x}(t)$ is replaced by a sequence of discrete states $\{\mathbf{x}_d\}$ in the same manner as discretized trajectories are generated by numerical integration (simulation) of dynamical systems. An optimal solution has to fulfill the discretized differential equations (and additional constraints) and it is optimal with respect to the discretized cost functional, i.e. it is a solution to a nonlinear optimization problem and approximates the solution of the original OCP.

#### 5.3.6.1 The Direct Optimal Control Technique DMOC

DMOC (*Discrete Mechanics and Optimal Control*, [151]) is a direct optimal control method tailored to the special structure of mechanical systems . The forced Euler-Lagrange equations (5.17b) are derived from a variational principle: the Lagrange-d'Alembert principle ([140]). DMOC is based on a direct discretization of the Lagrange-d'Alembert principle of the mechanical system. The goal of this discrete variational mechanics approach is to derive discrete approximations of the solutions of the forced Euler-Lagrange equations that inherit the same qualitative behavior as the continuous solution. For the discretization, the state space $TQ$ is replaced by $Q \times Q$ and the discretization grid for the time interval $[0,T]$ is defined by $\Delta t = \{t_k = kh \,|\, k = 0,\dots,N\}$, $Nh = T$, where $N$ is a positive integer and $h$ is the step size. The path $\mathbf{q} : [0,T] \to Q$ is replaced by a discrete path $\mathbf{q}_d : \{t_k\}_{k=0}^{N} \to Q$, where $\mathbf{q}_k = \mathbf{q}_d(kh)$ is an approximation of $\mathbf{q}(kh)$ [141, 151]. Similarly, the control path $\mathbf{u} : [0,T] \to U$ is replaced by a discrete one. The discrete Lagrange-d'Alembert principle then leads to the **discrete forced Euler-Lagrange equations**

$$D_1 L_d(\mathbf{q}_k, \mathbf{q}_{k+1}) + D_2 L_d(\mathbf{q}_{k-1}, \mathbf{q}_k) + \mathbf{f}_k^- + \mathbf{f}_{k-1}^+ = 0 \tag{5.18}$$

for each $k = 1,\dots,N-1$, where $D_i$ denotes the derivative w.r.t. the $i$-th argument. That means, solution curves of the differential equation (5.17b) can be approximated by discrete solution trajectories of the set of algebraic equations. In other words, for given control values $u_k$, equation (5.18) provides a time stepping scheme for the simulation of the mechanical system which is called a variational integrator (cf. [141]). Since these integrators, derived in a variational way, are structure-preserving, important properties of the continuous system are preserved (or change consistently with the applied forces), such as symplecticity or momentum maps induced by symmetries (e.g. the linear or angular momentum of a mechanical system). In addition,

[12] Confere e.g. [140] for a general introduction into the theory of mechanical systems, in particular regarding Lagrangian mechanics.

[13] Simply speaking, a manifold is a generalization of the vector space $\mathbb{R}^n$ including e.g. tori, but readers non-familiar with differential geometry can replace $Q$ by $\mathbb{R}^n$ and $TQ$ by $\mathbb{R}^{2n}$ in the following.

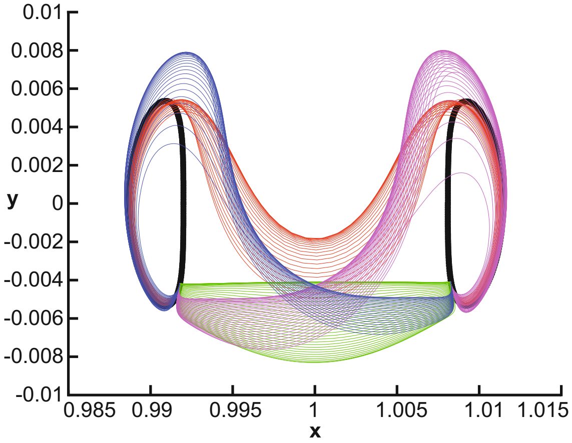


**Fig. 5.36** *Space mission design:* Pareto optimal trajectories for minimal control effort and time-minimal transfer between period orbits near sun and earth. Red: high mission times, low control effort. Green: small mission times, high control effort. Blue and magenta: solution between the first two (Figure from [152]).

their long-time energy behavior is excellent. Therefore, variational integrators can be used with relatively large step sizes. However, rather than solving initial value problems, an optimal control problem has to be solved, which involves the minimization of a cost functional $J(\mathbf{x},\mathbf{u}) = \int_0^{t_f} C(\mathbf{x}(t),\mathbf{u}(t))\,dt$. Thus, in the same manner, an approximation of the cost functional generates the discrete cost functions $C_d$ and $J_d$, respectively. The resulting nonlinear restricted optimization problem reads

$$\min_{\mathbf{q}_d,\mathbf{u}_d} J_d(\mathbf{q}_d,\mathbf{u}_d) = \min_{\mathbf{q}_d,\mathbf{u}_d} \sum_{k=0}^{N-1} C_d(\mathbf{q}_k,\mathbf{q}_{k+1},\mathbf{u}_k) \tag{5.19}$$

subject to the discrete forced Euler-Lagrange equations (5.18) together with discretized boundary and (in-)equality constraints for states and/or controls. Thus, the discrete forced Euler-Lagrange equations serve as equality constraints for the optimization problem which can be solved by standard optimization methods like SQP (cf. e.g. [76]). In [151], a detailed analysis of DMOC resulting from this discrete variational approach is given. The optimization scheme is symplectic-momentum consistent, i.e. the symplectic structure and the momentum maps corresponding to symmetry groups are consistent with the control forces for the discrete solution independent of the step size $h$. Thus, the use of DMOC leads to a reasonable approximation of the continuous solution, also for large step sizes, i.e. a small number of discretization points. Furthermore, constraints of mechanical systems can be included in DMOC such that it is applicable to constrained systems, which often occur in multi-body dynamics ([134]).

### 5.3.6.2 Extensions and Applications

Typically, in particular for self-optimizing systems, there is more than one single objective that has to be optimized, hence we are faced with **multiobjective optimal control** . Problems of this kind, i.e. with a vector $\mathbf{J}(\mathbf{x},\mathbf{u}) = (J_1(\mathbf{x},\mathbf{u}),\ldots,J_m(\mathbf{x},\mathbf{u}))$

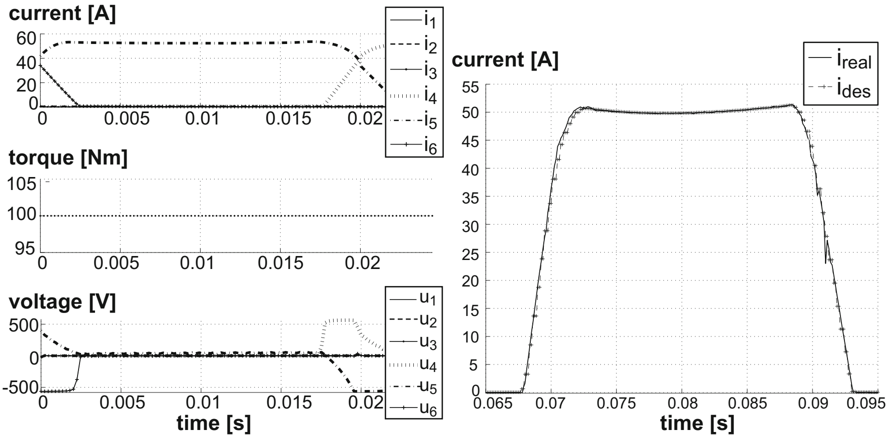


**Fig. 5.37** *Switched reluctance drive:* Optimal profile for current and voltage computed by DMOC. Note that the constraint of a fixed motor torque is fulfilled for every discretization point. The profile is combined with a feedback controller. It can be followed at the real test bench very well as shown in the right plot (original Figure from [63])

of cost functionals can be solved by a combination of multiobjective optimization methods and optimal control techniques. Since the discretization of the differential equations, e.g. by DMOC as described above, leads to a high-dimensional multiobjective optimization problem (i.e. a high number of optimization parameters $\mathbf{q}_d, \mathbf{u}_d$), image space oriented methods should be applied. In [152], this method has been applied to an optimal control problem in space mission design, cf. Fig. 5.36. Here, the concurring objectives are the control effort and the transfer time, which should both be simultaneously minimized. Thus, the solution of the multiobjective optimal control problem results in a number of very different Pareto optimal trajectories. A mission designer would now choose one of the correspondent control trajectories dependent on current aims and restrictions on the mission for a thorough analysis and further optimization with more detailed models.

As proposed above, the DMOC method is not restricted to purely mechanical systems since many electrical (sub)systems can be modeled by Lagrangian functions as well . In the course of the CRC 614, the optimization of the Hybrid Energy Storage System (cf. Sect. 5.3.1 and Sect. 2.1.5) has been repeated with additional (final) constraints on the optimal control problem. Furthermore, DMOC has been successfully used for the optimal control of a switched reluctance drive (cf. [63] and Sect. 2.1.1 above for a description of the test bed). The optimal current profiles have to fulfill two aims: maximizing the efficiency of the engine and guaranteeing a constant torque of the drive. In this application, the torque restriction is modeled as an equality constraint and the resulting single objective optimization problem is solved by DMOC. The resulting feedforward control is combined with a feedback controller. In Fig. 5.37, results are shown from the successful application to the real test bed.

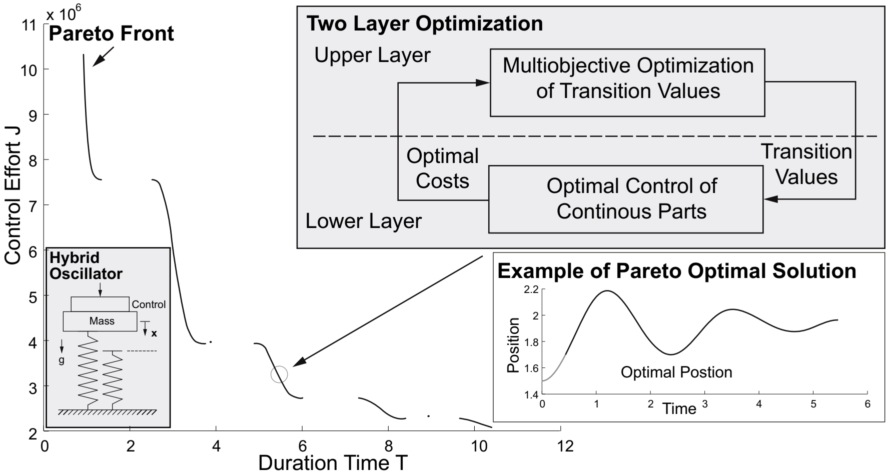


**Fig. 5.38** *Hybrid single mass oscillator:* in the two layer optimization approach, a multiobjective optimization problem arises since both the control effort and the time of the maneuver (into the equilibrium position) have to be optimized. The resulting Pareto front is shown with an example solution for the resulting hybrid position trajectory (Subimages from [61])

**Hybrid mechanical systems** are described by continuous-time dynamics in combination with discrete events to model e.g. impacts, varying topologies of interacting robots, or a changing environment. From the perspective of optimal control, the switching times at which the discrete events occur, become new design variables. The **optimal control of hybrid systems** is an active field of research. Promising results can be achieved by approaches that split the problem up into several layers (cf. Fig. 5.38 and [61] for a detailed discussion). It is then possible to solve ordinary optimal control problems on a lower layer with well established methods while on the upper layer, the switching time optimization can be performed with other appropriate techniques. Figure 5.38 shows an example with a multiobjective optimization of a hybrid single mass oscillator, which has to be steered into its equilibrium position. Switching time optimization as a specific, isolated optimal control problem for hybrid systems has been studied in [60].

The optimal control method DMOC has been extended in several directions to improve performance and applicability even further. For the computation of gradients that are used for the optimization, e.g. by the SQP algorithm, DMOC can be combined with ADOL-C [153], a tool for algorithmic differentiation. In applications where subsystems with different time scales are interacting, the variational integrator can be extended to a multirate integration scheme [132, 133] that allows for an accurate integration with acceptable computational effort for combined fast and slow dynamical systems. The accuracy of numerical integration and thus of optimal solutions as well depend on the order of the approximation scheme. Therefore, higher order schemes can be used for the discrete Lagrangian [34].

Direct optimal control methods are based on local optimizers for the nonlinear optimization problem and therefore, they strongly rely on good initial guesses. Since minimal control effort is often a desired aim, it is a fruitful approach to use **inherent dynamical properties** of the uncontrolled system to generate such initial guesses. In space mission design, it has become state of the art to use trajectories on the system's invariant manifolds to design energy efficient control maneuvers (cf. e.g. [45, 146, 199] for applications using DMOC as the optimal control method). This approach can be used for technical systems as well, in [64] it is shown that, compared to black box optimizations with simple initial guesses, better (local) optima can be found with the help of initial guesses on the stable manifold of the final equilibrium position for a planar double pendulum. In more detail, this idea is explained in the broader context of motion planning with motion primitives in the following section.

### *5.3.7 Motion Planning with Motion Primitives*

Kathrin Flaßkamp and Sina Ober-Blöbaum

Solving optimal control problems which arise in real applications is a challenging task for current numerical techniques. Since many optimal control techniques are based on local optimization methods, they strongly depend on good initial guesses to provide (local) optimal solutions which are also globally efficient. **Motion planning with motion primitives** – going back to [66] – tackles these difficulties with a two phase approach. In the first step, several short pieces of simply controlled trajectories are collected in a motion planning library, typically represented as a graph. These motion primitives can be sequenced to longer trajectories in various combinations. In the second phase, for a given optimal control problem, the optimal sequence of motion primitives is determined from the motion planning library. Such motion primitives originate from inherent **symmetries**, i.e. the dynamical system is equivariant with respect to certain transformations and certain system properties turn out to be invariant with respect to these symmetries[14]. Typically, mechanical systems naturally exhibit symmetries as translational or rotational invariance. By consequence, controlled maneuvers, that have been computed for a specific situation, are suitable in many different (equivalent) situations as well. Recently (cf. [62]), this motion planning technique has been extended by a new kind of primitives, namely trajectories on (un)stable manifolds of the natural system dynamics. In space mission design, such trajectories on invariant manifolds have already been successfully used (cf. e.g. [146]). This approach is especially tailored to the computation of energy efficient (minimal control effort) solutions, which is often a major objective for technical systems.

We formally introduce symmetry and motion primitives for Lagrangian systems (cf. 5.3.6), although the basic approach holds for general dynamical systems (cf. [66]). Assume that a Lie group $G$ is acting on the configuration manifold $Q$ by

[14] Again, we recommend [140] for an introduction to the role of symmetries in mechanical systems.

a so called left-action $\Phi : G \times Q \to Q$ ($\Phi(\mathbf{g},\cdot) =: \Phi_{\mathbf{g}}$ is a diffeomorphism for each $\mathbf{g} \in G$). It can be lifted to the tangent space: $\Phi^{TQ} : G \times TQ \to TQ$ for $(\mathbf{q},\mathbf{v}) \in TQ$ given by $\Phi_{\mathbf{g}}^{TQ}(\mathbf{q},\mathbf{v}) = T(\Phi_{\mathbf{g}}) \cdot (\mathbf{q},\mathbf{v})$. Then, symmetry corresponds to the invariance of the Lagrangian under the group action, i.e. $L \circ \Phi_{\mathbf{g}}^{TQ} = L$ for all $\mathbf{g} \in G$. In other words, two trajectories $\pi_1 : t \in [t_{i,1}, t_{f,1}] \mapsto (\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{u}_1)(t)$ and $\pi_2 : t \in [t_{i,2}, t_{f,2}] \mapsto (\mathbf{q}_2, \dot{\mathbf{q}}_2, \mathbf{u}_2)(t)$ are equivalent, if it holds that (1) $t_{f,1} - t_{i,1} = t_{f,2} - t_{i,2}$, both have the same time duration and (2) there exists $g \in G, T \in \mathbb{R}$, such that $(\mathbf{q}_1, \dot{\mathbf{q}}_1)(t) = \Phi_{\mathbf{g}}^{TQ}((\mathbf{q}_2, \dot{\mathbf{q}}_2)(t-T))$ and $\mathbf{u}_1(t) = \mathbf{u}_2(t-T) \, \forall t \in [t_{i,1}, t_{f,1}]$. All equivalent trajectories can be summed up in an equivalence class, the motion primitive. The number of candidates for the motion planning library can be immensely reduced by exploiting the system's invariance, i.e. only a single representative is stored that can be used at many different points when transformed by the lifted symmetry action. Induced by the symmetry, trim primitives are a special class of motion primitives. They are constantly controlled solutions which are generated solely by the symmetry action, i.e. $(\mathbf{q}, \dot{\mathbf{q}})(t) = \Phi^{TQ}(\exp(\xi\, t), (\mathbf{q}_0, \dot{\mathbf{q}}_0)), \mathbf{u}(t) = \mathbf{u}_0 = \text{const.} \, \forall t \in [0, T]$ with $\xi \in \mathfrak{g}$, the corresponding Lie algebra and $\exp : \mathfrak{g} \to G, \xi \mapsto \exp(\xi) \in G$ (cf. [140] for an introduction to mechanical systems and symmetry from a differential geometric perspective). Trim primitives can be found analytically or numerically based on the symmetry action. For mechanical systems, they are identical to relative equilibria and can be computed by symmetry reduction procedures (cf. [62]).

The second type of primitives, trajectories on (un)stable manifolds are computed for fixed points (or equilibria) $\bar{\mathbf{x}} = (\bar{\mathbf{q}}, 0)$ of the uncontrolled system. The local stable manifold for a neighborhood $U$ of $\bar{\mathbf{x}}$ is defined as $W_{loc}^{s}(\bar{\mathbf{x}}) = \{\mathbf{x} \in U \,|\, \mathbf{F}_L(\mathbf{x}, t) \to \bar{\mathbf{x}} \text{ as } t \to \infty \text{ and } \mathbf{F}_L(\mathbf{x}, t) \in U \, \forall t \geq 0\}$. Then, the global stable manifold can be obtained by the union of the (pre)images of the Lagrangian flow $\mathbf{F}_L$. A stable manifold consists of all points in state space flowing towards the equilibrium. The corresponding trajectories are promising candidates for energy efficient steering maneuvers to operation points which are often the fixed points. The unstable manifold consists of all points that show the same behavior in backward time. Their existence is guaranteed by the stable manifold theorem [84]. In general, the (un)stable manifolds have to be computed numerically, e.g. by set-oriented methods [44].

As a third class of primitives, short controlled maneuvers between trims and manifold trajectories are required such that the primitives can be sequenced. They can be computed by DMOC (cf. Section 5.3.6) for example. The computed primitives are stored in a library. Then, for a specific control problem, i.e. with initial and final points on trims, e.g. operation modes of mechanical systems, in the library it is searched for the optimal sequence of primitives. This can be done based on the graph representation, the so called **maneuver automaton** (cf. [62, 66]). In principle, this second step could be even performed in real time, when using appropriate graph search methods.

We illustrate the approach for a spherical pendulum. Its Lagrangian is given by $L(\varphi, \dot{\theta}, \dot{\varphi}) = \frac{1}{2} m r^2 (\dot{\varphi}^2 + \dot{\theta}^2 \sin^2(\varphi)) - mgr(\cos(\varphi) + 1)$ and we assume forcing in both directions. The system is symmetric with respect to rotations about the vertical axis. Trims are horizontal rotations with constant velocity. Contrarily, the (un)stable manifolds of the upper equilibrium are purely vertical motions. For an example

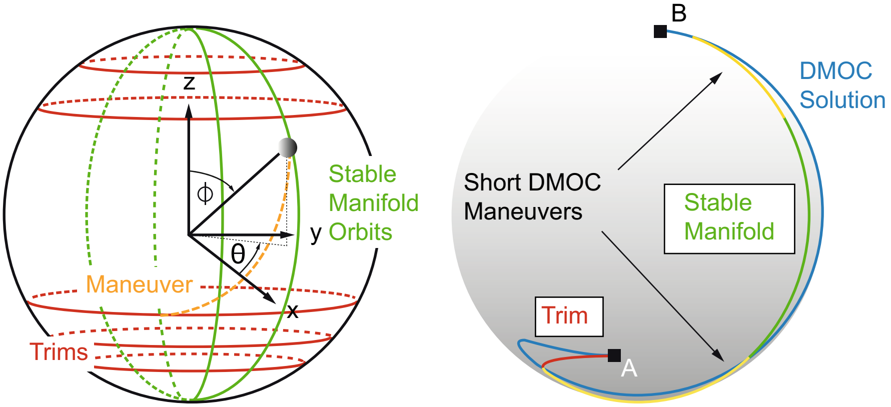


**Fig. 5.39** *Sphercial pendulum:* in the motion planning with motion primitives approach, trim primitives are horizontal rotations while orbits on manifolds are purely vertical motions. An optimal sequence has been computed by DMOC and used for a post optimization ("DMOC solution", original subimages from [62])

control problem, the resulting optimal sequence is shown in Fig. 5.39 that consists of five motion primitives: the initial and final trim, two connecting maneuvers, and a trajectory on the stable manifold in between. The sequence has been used for a post-optimization by DMOC.

### *5.3.8 Hierarchical Hybrid Planning*

Bernd Kleinjohann, Lisa Kleinjohann, and Christoph Rasche

Hybrid Planning [3] is based on hierarchical modeling presented in Sect. 5.3.3 and Pareto points calculated by a multiobjective optimization presented in Sect. 5.3.5. Taking the RailCab system (cf. Sect. 2.1) into account, several constraints have to be considered when moving a single RailCab from an initial position to a given goal position. One constraint is that the RailCab has only limited energy resources. To take such constraints concerning discrete as well as continuous system parameters into account, an overall plan for the movement of the RailCab must be computed. The term hybrid planning [2] denotes the integration of discrete and continuous domains in the planning approach. Initially a plan is created offline. It is updated continuously during the movement of the RailCab to ensure that, e. g. environmental influences, like wind do not lead to a violation of the given constraints. Hierarchical hybrid planning [56] denotes a planning approach, which does not only combine discrete and continuous planning but also considers the system's hierarchical decomposition into its single parts (cf. Sect. 5.3.3) during planning.

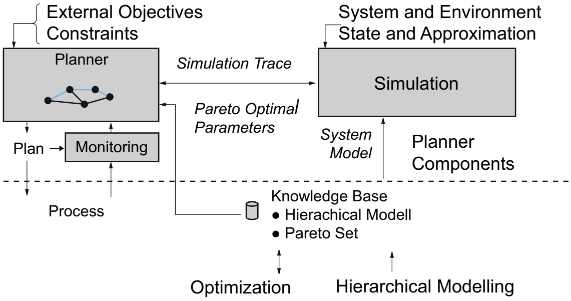


**Fig. 5.40** Architecture of the hierarchical hybrid planning system

### 5.3.8.1 Principle

A plan is computed in order to ensure that a RailCab moves from its initial position to its destination while the requirements are taken into account by the planner. Different parts of a traveling route have diverse properties, like, e. g. slopes which have to be modeled. Thus, to actually create such a plan the complete route between the initial and the end position of the RailCab is subdivided into single track segments. In the first step an initial plan is computed consisting of different parameter settings for the single parts of the system for each track segment. These parameter settings build a discrete dimension of choice for the planner. In the case of the RailCab system considered here several objectives regarding for instance values like passenger comfort and energy consumption, which are in conflict have to be taken into account. For handling such conflictive objectives a multiobjective Pareto optimization is used. The Pareto optimization calculates a Pareto front determining optimal trade offs between parameter settings for each track section of the selected traveling route. Then, the offline planner selects a single Pareto point from the Pareto front for each section. Due to the actual system or environment conditions like wind, abrasion, etc. the forecasted results which selected parameter settings of the plan should lead to, might not be reached. Such deviations between the plan and the actual conditions are detected by continuously monitoring several values that determine the system state, allowing to initiate replanning by the online planner. Fig. 5.40 shows the components used to implement this approach.

As described in detail in [56] the planner is equipped with overall external objectives that need to be fulfilled at any time. In order to forecast the future development of continuous values determining the system state, the planner initiates a simulation with the actually measured system and environment state for the considered actions. The result of the simulation is a number of continuous value traces that are evaluated according to the constraints and objectives. The constraints are used to rule out an action, e.g. if the maximum peak power is too high or the comfort value used by the system is too poor during the simulation. Otherwise the action is considered as possible alternative by the planner. To finally decide for a possible alternative it is further evaluated with respect to the external objectives, for instance regarding the mean comfort or energy consumption of the overall section.

### 5.3.8.2 Methodology

One important issue of a hierarchical hybrid planner is the computation of a multilevel hierarchical configuration of system parameters during system operation. This configuration is used to improve the offline plan to take additional constraints or external objectives into account when different environment conditions or track segment properties must be considered.

The planner takes its input from the hierarchical optimization (cf. Sect. 5.3.2) and the hierarchical modeling (cf. Sect. 5.3.3). The results of these components are abstract models of the system parts and a set of Pareto-optimal parameter settings. As these components are designed in a hierarchical fashion, the outputs are precalculated on different hierarchical levels.

As an example for illustrating the methodology the Active Suspension Module of the RailCab (cf. Sect. 2.1.4) may serve. The active suspension system of the RailCab can be partitioned and structured hierarchically according to the function of each module. The hierarchy consists of two levels. On the upper level, the entire system which is in charge of the active suspension is considered. Beneath, on the lower level, there are two actuator groups realizing the active suspension by ensuring correct deflections of the fiberglass reinforced polymer springs (cf. Sect. 2.1.4).

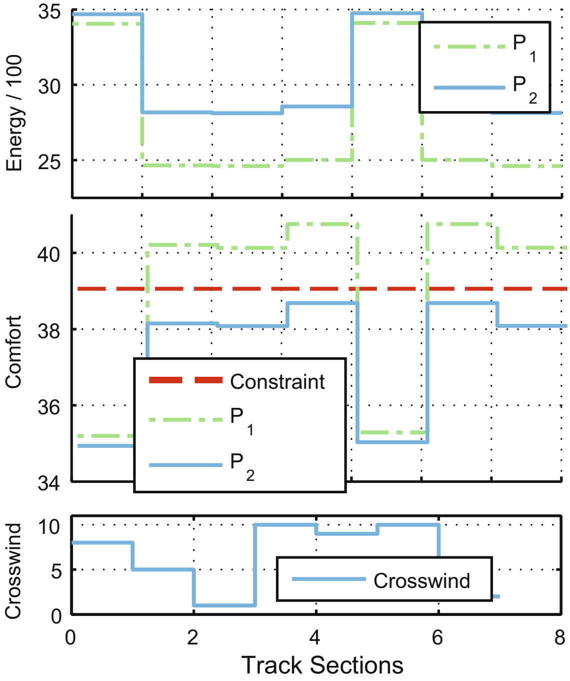
The planner computes a multilevel configuration of the parameters of the active suspension system for each track segment based on the inputs described before. If the constraints could not be met, the planner adjusts only the lower level settings to reach the current goals, without affecting the upper level settings. Different objectives can be handled by different hierarchical levels.

During the movement of a RailCab, which executes a given plan, a monitoring of the current system behavior by measuring values like energy consumption and given comfort takes place. The measured data is compared to the data, which were taken into account to compute the initial plan. If the difference between this data is too high, a replanning is necessary. Hence, during system movement alternative Pareto-optimal parameter configurations have to be selected, which take into account these deviations. For this purpose the simulation component is used to predict the system behavior resulting from alternative parameter settings for the next track sections. These settings build a discrete dimension of choice for the planner and can be used for a predictive planning of the next track sections.

### 5.3.8.3 Application and Evaluation

The approach was evaluated using the Active Suspension Module (cf. Sect. 2.1.4), which is a part of the RailCab. The values for energy and comfort are abstract values without units of measurement. The test track consists of seven track sections. An overall energy consumption with the value 10600 and a comfort constraint for each track section of 49 was given. The two constraints, energy consumption and comfort, are in conflict because a higher comfort leads to a higher energy consumption. Two different types of planning were compared. They also considered the influence of changing environmental conditions, in this case represented by varying crosswind

**Fig. 5.41** Results of a test run. As higher the energy consumption is as higher is the value. A higher comfort is represented by a lower value.



settings. First, a hybrid planning was performed, which led to a resulting plan $P_1$. Thereafter, the hierarchical hybrid planning approach was executed using the same constraints on the same test track. While the non-hierarchical approach was not always able to reach the constraints, the hierarchical approach computed a plan $P_2$ in which each constraint was always reached with only small increases in energy consumption. A more detailed evaluation of the results is represented in [56].

The results in Fig. 5.41 show the energy values and comfort values as well as the crosswind settings on each track segment.

The results show that only the hierarchical approach was able to consistently meet the comfort constraints by finding a feasible plan. The non hierarchical planner violated the constraints at the track segments $2-4$ and $6-7$. The reason is, that the non-hierarchical planner did not have enough options to consider these constraints. The hierarchical planner can change the Pareto points influencing the lower level parts, i.e. the two actor groups realizing the active suspension as mentioned above. In contrast, the non-hierarchical planner can only change to another Pareto point for the overall system in order to satisfy the given constraints. This leads to different and sometimes inadmissible configurations. The concrete values selected by the planners $P_1$ and $P_2$ are shown in Table 5.2.

Table 5.2 shows that changes on the upper level have a much higher effect on the resulting values than changes on the lower level. Changes on the lower level also have an effect when both planners choose the same Pareto points for each track segment on the upper level. Only the hierarchical approach was able to meet the constraints due to its ability to change the Pareto points influencing the lower level parts.

The results show that the hierarchical planning can use a more precise configuration to match the given constraints. Nevertheless, improvements regarding one parameter always imply impairments regarding other conflictive parameters, due to the Pareto-nature of the available planning alternatives.

**Table 5.2** Energy and comfort values. Maximum of Total Energy 28.300. Comfort Constraint 39

| | Energy | | Comfort | |
|---|---|---|---|---|
| | P1 | P2 | P1 | P2 |
| *section*1 | 4812.4 | 4940.23 | 35.14 | 34.87 |
| *section*2 | 2930.19 | 3634.37 | 40.15 | 38.09 |
| *section*3 | 2920.29 | 3625.22 | 40.07 | 38.02 |
| *section*4 | 3001.84 | 3714.25 | 40.7 | 38.62 |
| *section*5 | 4823.7 | 4952.7 | 35.23 | 34.97 |
| *section*6 | 3001.84 | 3714.25 | 40.7 | 38.62 |
| *section*7 | 2920.83 | 3625.69 | 40.08 | 38.02 |
| *amount* | 24411.09 | 28206.71 | 272.07 | 261.21 |

## 5.3.9 Statistical Planning

Bernd Kleinjohann, Lisa Kleinjohann, and Christoph Rasche

Taking statistical data into account to compute plans for mechatronical systems results in a self-optimizing behavior. This is due to the fact that observations of previous system behavior are used to improve the so called policy describing the action selection strategy of the system. This principle of learning from observations is used to construct an intelligent self-optimizing system that is able to fulfill several predefined tasks in dynamically changing environments.

### 5.3.9.1 Principle

The statistical planning approach for mechatronic systems described in this section relies mainly on a statistical data base and rewards; it applies the principles of Reinforcement Learning. The environment is modeled as a discrete finite non deterministic Markov decision process as described by Sutton and Barto (1998) [197]. The mechatronic system measures its current state, selects an action according to a given policy and performs this action. This leads to a transition into a new state and generates a reward signal. Finally, the mechatronic system observes the new state and the reward and compares it with the previous state and the action performed. Based on this comparison the policy is adapted. The objective is to maximize the reward.

#### 5.3.9.2 Methodology

One requirement is that the statistical planning, i. e. the creation of statistical data and the planning, must be done online. This requirement arises since it is hard to create a statistical data base for a real world system where model values appropriately reflect the properties of the environment in which the system works. Hence, the algorithms used for statistical planning must take into account the limited computational power of the mechatronic system.

One algorithm recently investigated by several researchers, which is able to fulfill the requirements is **Q-Learning** [205]. It is an off-policy temporal difference learning algorithm and uses an action-value-function whose update can be expressed recursively. This allows the online execution of the algorithm. The action-value-function $Q(s,a)$ is used to compute the benefit, if a given action in a given state is executed while a fixed policy follows. This action-value-function is similar to the cost functionals presented in Sect. 1.4.1.

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \left[ r_{t+1}(s,a) + \gamma \max_{a'} Q_{t+1}(s',a') - Q_t(s,a) \right] \tag{5.20}$$

Equation 5.20 is a so called sample backup update of the action value function where $r(s,a)$ specifies the reward for taking action $a$ in state $s$, $\alpha$ denotes a step size parameter and $\gamma$ a discounting factor used to handle continuous tasks. The step size parameter controls the learning rate while the discounting factor determines the importance of future rewards. $Q(s,a)$ is the quality of a state-action combination. One drawback when using this approach for statistical planning is that it needs a large number of episodes before it converges making it very time consuming. An episode is a single run from the initial configuration until $s'$ is a final state.

To overcome this problem, the **Prioritized Sweeping** algorithm [147] can be used. The main idea is that a model of the environment is maintained. In this context the term model means everything the mechatronic system can use to predict the reaction of the environment when a certain action is performed while the system is in a certain state. In the described case it means that after the performance of an action and updating the action value function, several steps are simulated using the stored model of the environment to predict the outcome. This can speed up the approach by a factor of several magnitudes compared to the classical Q-Learning approach. Additionally, convergence to the optimal policy can be guaranteed, as shown by Li and Littmann (2008) [136].

#### 5.3.9.3 Platforms and Applications

The approach was implemented on the miniature robot platform BeBot [99] (cf. Sect. 2.2) in order to improve the behavior for the application presented in Sect. 2.2.4. The main sensor used to measure the current state is the camera of the BeBot. Moreover, in order to use classical learning algorithms like Q-Learning, a few assumptions concerning the state space of the environment were made. The state space is assumed to be discrete and of finite size. In addition, a restriction of the set of possible actions took place and the mapping of abstract actions to the actual motor commands was fixed. The used state space and action set is based on Asada et al. (1995) [11].

Image processing is done directly on the BeBot using a color based image segmentation and feature classification approach [111]. The data extracted from the images are then used to determine the current state of the BeBot, for instance, its (discretized) distance to objects in its environment, which could either be goals it has to reach or obstacles it has to avoid. This state information is further used as input of the behavior module, i. e. as input of the statistical planning algorithm.

Practical evaluations of this approach revealed, that noisy images and high sensitivity of the color based feature extraction to illumination changes often lead to the detection of abrupt state changes of the system, e.g. since the detected objects or their positions vary between subsequent images. Another problem is that using the camera a BeBot is not able to perfectly determine its current state. Often several states can be possible due to the limited information the BeBot receives through the use of its camera. The problems were solved by considering a so called hidden state in the model. These models are called partially observable Markov decision process (POMDP) [143]. In POMDPs it is assumed that the actual state (hidden state) of the underlying Markov decision process is not directly observed but the given observations appear with a certain probability in each state. Rather than always having a fully observable state, a belief state probability distribution over all the states has to be maintained. The probability for each state $s$ to be the belief state can be computed recursively, i. e. based on the last belief state, the last action $a$, the current observation $o$, and the transition and emission probability parameters of the model as shown in Eq. (5.21).

$$b^{t+1}\left(s_j\right) = \Pr\left(s_j|o,a,b^t\right) = \frac{\Pr\left(o|s_j\right)\sum_{s_i\in S}\Pr\left(s_j|s_i,a\right)b^t\left(s_i\right)}{\sum_{s_k\in S}\Pr\left(o|s_k\right)\sum_{s_i\in S}\Pr\left(s_j|s_i,a\right)b^t\left(s_i\right)} \tag{5.21}$$

The parameters of the model can be computed offline using a modified version of the Baum-Welch algorithm [143].

This leads to a belief state which is no longer discrete. That makes it impossible to find an optimal policy using the described algorithms. Solving POMDPs directly needs a high computational effort. So, only the most likely state and output are considered to be the actual hidden state. The BeBot then takes the selected state as its current state and uses it as the basis to determine its next action. Based on this method only the underlying MDP must be solved using Q-Learning or Prioritized Sweeping as described above.

### 5.3.10 Behavior Planning in Nondeterministic Environment

Philip Hartmann

In order to increase the dependability of self-optimizing mechatronic systems, **cognitive planning components** with enhanced information processing are also integrated into the system. These components allow mechatronic systems to plan their behavior in order and fulfill individual tasks independently and proactively. A single task represents a sequence of actions executed by the mechatronic system within a limited time frame in order to reach a given goal state. Along with bare fulfillment of that task, i.e. finding an arbitrary sequence of actions to reach the desired goal-state, planning tries to minimize or maximize objectives, such as minimizing energy consumption. For this reason, actions are only selected if their expected results fit the desired objectives. With respect to dependability, it is possible to create alternative plans for critical situations before they arise, i.e. for particular environmental or low energy situations. However, this may decrease the availability of the mechatronic system and the reliability of subsequent task fulfillment. Furthermore, behavior planning considers the continuous and nondeterministic environment of the system (cf. [118]).

When modeling a planning domain for behavior planning of intelligent mechatronic systems (cf. [118, 119, 125]), the main challenge is to map the partial function solutions onto actions within the framework of PDDL (Planning Domain Definition Language, cf. [65]). Depending on the amount of detail desired when modeling these functions, this approach results in a higher or lower abstraction of actions. In case the of behavior planning, the executed partial function solutions are called operation modes. Thus, a planning problem for mechatronic systems can be formulated as follows (adapted from [119]):

- $OM$ is a finite set of available operation modes,
- $S$ is a finite set of possible system states, and
- $\mathbf{s} \in S$ is a state vector with $s(i) \in \mathbb{R}$ for the i-th component.

Furthermore, for each operation mode $om \in OM$:

- $prec^{om} := \{(x_{lower} < s(i) < x_{upper}) | x_{lower}, x_{upper} \in \mathbb{R}\}$ is the set of preconditions which must be true for the execution of operation mode $om$ and
- $post^{om}$ is a set of conditional numerical functions describing the change of influenced state variables. A condition is a logical expression (conjunctions and disjunctions) of comparison operations; if a condition is true, the result of the corresponding numerical function is assigned to state variable in the next state $\mathbf{s}'$ of the plan [119].

A specific planning problem is the finding of a sequence of operation modes which describes a transition from an initial system state $\mathbf{s}_i \in S$ to a predetermined goal state $\mathbf{s}_g \in S$. Thus, a single task of a mechatronic system is given as a 2-tuple $O = (\mathbf{s}_i, \mathbf{s}_g)$. A solution to the planning problem can be determined by applying a state space search algorithm (cf. [74]), for example. The optimal solution (e.g. minimum of energy consumption) can be found by computing the specific solutions with respect to the given System of Objectives. For this purpose, $\Omega$ is a set of objectives and $f : S \times \Omega \rightarrow [0,1]$ is a function that indicates how well the execution of an operation mode in a given state satisfies the objective. Using the weighted sum of the objectives, the optimal sequence of operation modes can be determined (cf. [119]).

During runtime in a non-deterministic environment with continuous processes, behavior planning has to include methods for handling resulting problems. For example, Klöpper (2009) (cf. [118]) uses a modeling approach to integrate continuous processes based on optimal control and continuous multiobjective optimization (also cf. [73]), as well as estimation obtained by fuzzy approximation. To manage planning under uncertain conditions, different techniques can be combined in a hybrid planning architecture (cf. [119]).

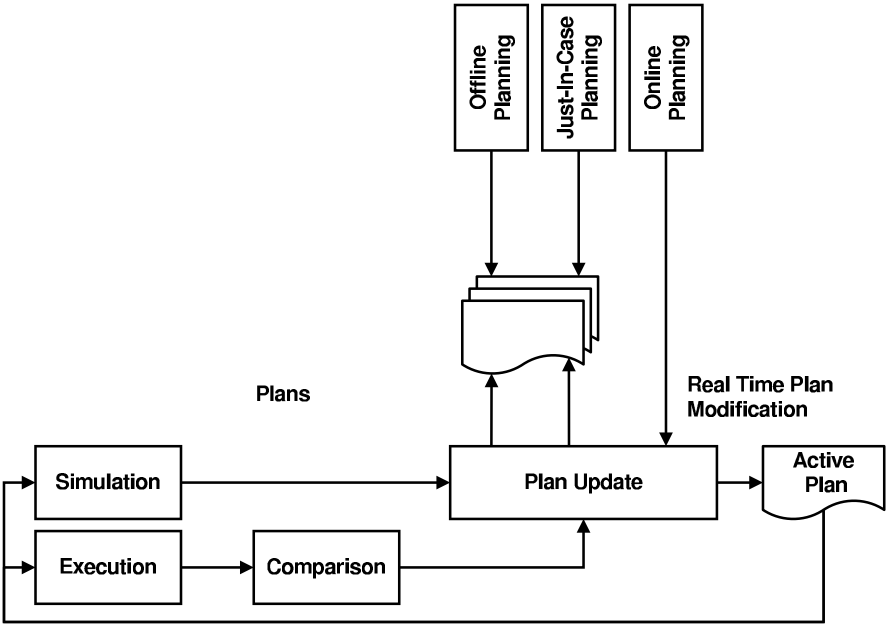


**Fig. 5.42** Hybrid planning architecture (source: [119])

Figure 5.42 shows the hybrid planning architecture with the corresponding components for planning, execution and monitoring of plans. The total planning is divided into three separate sections: offline, just-in-case and online planning. The offline planning represents a planning process where, initially, a deterministic and optimal plan in view of the objectives is fully created before execution. The resulting plan is used in the just-in-case planning to do a probabilistic analysis for plan deviations. The present and deterministic plan is examined for estimated variances in order to proactively generate conditional branches, with alternative plans for critical system conditions. A threshold specifies the maximum probability of state deviations which would result in a generation of conditional branches (see [125], in particular also [125] and [118].)

For this purpose, an additional stochastic planning model is formulated based on the deterministic planning model. This consists of stochastic states $\mathbf{s}^p$ with $|\mathbf{s}^p| = |\mathbf{s}|$, where $range(s^p(i)) \rightarrow P(\mathbb{R})$ is the values range and $distribution(s^p(i))$ the probability distribution of the state variable $s^p(i)$ and a stochastic variant of the operation modes. Let $in_s^{om} \subseteq pre^{om}$ be a subset of input variables and $out_s^{om} \subseteq post^{om}$ a subset of output variables. For each output variable $o \in out_s^{om}$, a Bayesian network (cf. [20]) $bn_o^{om}$ is created to formulate the stochastic relation (cf. [118, 119]; for a concrete example of creating a stochastic model cf. [125]). As a result, it is now possible to use the just-in-case-planning to generate alternative plans for situations that could occur with high probability during operation.

The online planning (cf. Fig. 5.42) serves as a fallback mechanism; it selects the optimal operation mode for the next execution step. Thus, operation in previously unplanned situations is guaranteed. A simulation of the continuous system behavior will check whether the current action of the active plan is executable under the given environmental conditions. If this is not possible, online planning is necessary, e.g. for a situation with extreme environmental influences such as heavy rain. While completing the execution of previously planned operation modes, a comparison of planned and actually reached system states is carried out.

A process for plan updating will check whether a pre-determined plan is available or whether a plan modification by the online planning is necessary. This will guarantee the immediate availability of the next operation mode (cf. Fig. 5.42).

The just-in-case and online planning are implemented as anytime algorithms (for the usage of anytime algorithms in intelligent systems cf. [214]). The planning process can be interrupted at any time to obtain a result, but with increasing time for calculations it provides a higher quality of result, as it is possible to generate more branches and to reach a higher depth of planning.

The dependability our type of system can be influenced by various factors. A major factor is the availability of energy, as this is crucial for the operation of the system. To ensure the dependability of the mechatronic system, it is essential to use the energy storage in a valid range and in particular to continuously observe the state of charge. Energy Management can use behavior planning to proactively schedule future energy demands according to the fulfillment of the current task, which increase the dependability of the mechatronic system (cf. [125]). Table 5.3 shows the values for operation modes derived from the multiobjective optimization from the Active Suspension Module of the RailCab system.

The experiments described here were intended to allow to evaluate three hypotheses (cf. [119]). One of these hypotheses in connection with the dependability was that a lower threshold probability and a higher number of alternative plans increases the reliability of the just-in-case planning ( [125]). The simulated experiments included four scenarios (source [119]):

**Table 5.3** Values for $f_1$ (weighted average body acceleration in $m/s^2$) and $f_2$ (energy consumption in ws) of operation modes derived from the multiobjective optimization of the Active Suspension Module. (source: [119])

| OM | Objective function | Track type | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | I | II | III | IV | V | VI | VII | VIII | IX | X |
| a | $f_1$ | 0.117 | 0.233 | 0.350 | 0.466 | 0.583 | 0.699 | 0.816 | 0.932 | 1.049 | 1.166 |
| | $f_2$ | 196 | 393 | 589 | 786 | 982 | 1179 | 1375 | 1572 | 1768 | 1965 |
| b | $f_1$ | 0.152 | 0.304 | 0.457 | 0.609 | 0.761 | 0.913 | 1.066 | 1.218 | 1.370 | 1.522 |
| | $f_2$ | 165 | 329 | 494 | 659 | 823 | 988 | 1153 | 1317 | 1482 | 1647 |
| c | $f_1$ | 0.192 | 0.385 | 0.577 | 0.770 | 0.962 | 1.155 | 1.347 | 1.540 | 1.732 | 1.925 |
| | $f_2$ | 142 | 283 | 425 | 567 | 709 | 850 | 992 | 1134 | 1275 | 1417 |
| d | $f_1$ | 0.224 | 0.449 | 0.673 | 0.897 | 1.122 | 1.346 | 1.570 | 1.794 | 2.019 | 2.243 |
| | $f_2$ | 122 | 245 | 367 | 489 | 612 | 734 | 856 | 979 | 1101 | 1224 |
| e | $f_1$ | 0.262 | 0.523 | 0.785 | 1.047 | 1.308 | 1.570 | 1.832 | 2.093 | 2.355 | 2.617 |
| | $f_2$ | 104 | 208 | 313 | 417 | 521 | 625 | 730 | 834 | 938 | 1042 |
| f | $f_1$ | 0.298 | 0.595 | 0.893 | 1.191 | 1.488 | 1.786 | 2.084 | 2.381 | 2.679 | 2.977 |
| | $f_2$ | 87 | 173 | 260 | 346 | 433 | 520 | 606 | 693 | 779 | 866 |
| g | $f_1$ | 0.331 | 0.662 | 0.994 | 1.325 | 1.656 | 1.987 | 2.318 | 2.649 | 2.981 | 3.312 |
| | $f_2$ | 69 | 138 | 206 | 275 | 344 | 413 | 482 | 550 | 619 | 688 |
| h | $f_1$ | 0.375 | 0.749 | 1.124 | 1.499 | 1.873 | 2.248 | 2.623 | 2.997 | 3.372 | 3.747 |
| | $f_2$ | 50 | 99 | 149 | 199 | 248 | 298 | 348 | 398 | 447 | 497 |
| i | $f_1$ | 0.435 | 0.870 | 1.305 | 1.739 | 2.174 | 2.609 | 3.044 | 3.479 | 3.914 | 4.349 |
| | $f_2$ | 27 | 55 | 82 | 110 | 137 | 164 | 192 | 219 | 247 | 274 |

1. ($\pm 0\%$): The energy consumptions drawn from track networks were not changed during simulation.
2. ($\pm 15\%$): The energy consumptions drawn from track networks were either decreased or increased by a random value up to $15\%$.
3. ($+15\%$): The energy consumptions drawn from track networks were always decreased by a random value up to $15\%$.
4. ($-15\%$): The energy consumptions drawn from track networks were always increased by a random value up to $15\%$.

The results are shown in Fig. 5.43 (for a detailed description of the simulation parameters and the executed scenarios cf. [119]) When regarding the percentage of failed plan execution during the simulation runs for different scenarios, adjusting the two parameters threshold values and number of alternative plans reduces the number of failed plans significantly.

A detailed explanation of behavior planning for mechatronic systems can be found in [119, 125]. In particular, [125] gives a deeper understanding of the probabilistic plan structure used in the analysis of the just-in-case planning. The basic methods were originally published in the dissertation by Klöpper (2009) [118], which may also be a good starting point for further information.

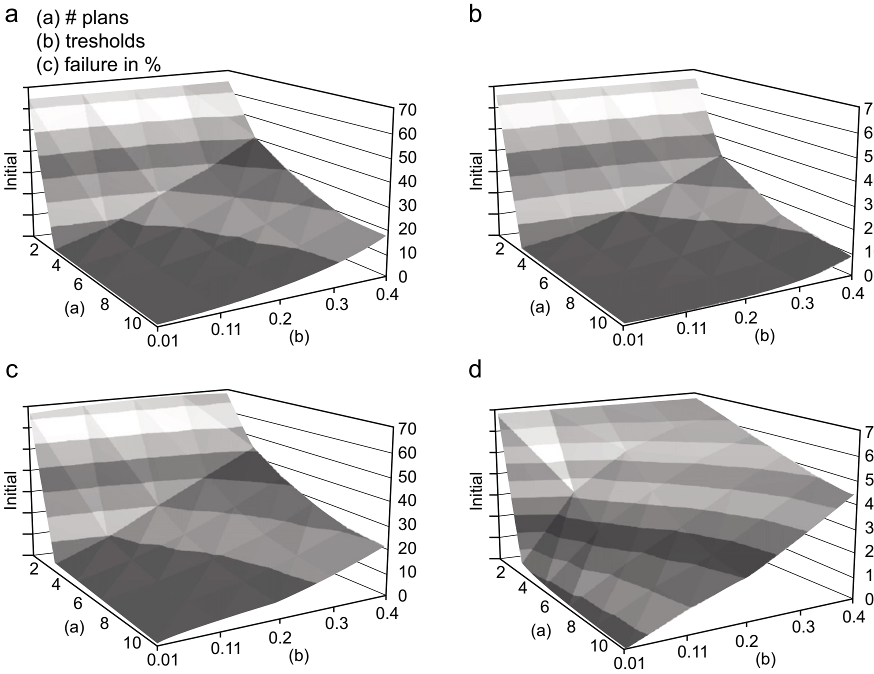


**Fig. 5.43** Percentage of failed execution depending on threshold probability and number of available alternative plans; (a) Return to Standardplan (±0%); (b) No Return to Standardplan (±0%); (c) Return to Standardplan (±15%); (d) Return to Standardplan (+15%) (source: [119])

## *5.3.11 FIPA Conform Cross-Domain Communication*

Philip Hartmann

Another advantage of self-optimizing systems is given by the possibility for intelligent communication of individual subsystems. The FIPA specifications (cf. [106]) provide a suitable way to implement cross-domain communication for autonomous mechatronic systems. To enable a more sophisticated approach the further considerations will include a requirement scenario for the RailCab system. A production facility is pursuing a just-in-time procurement strategy (JIT). To achieve this strategy, the transportation of goods is done by the RailCab system. For this purpose the production facility has access to a data base of RailCabs, which are able to deliver goods in the given time. Because of the RailCab's ability to work in a team, there is the possibility to take a closer look at complex voting scenarios to determine a suitable RailCab for specific orders. Both the production facility and the RailCab system are modeled as multi-agent systems with two different domains represented by ontologies (cf. [101]). The main goal of this section is to show a principal feasibility for the implementation of a FIPA based communication across the given domain boundary. First, an exemplary overview of the ontologies, that are available to the production facility (domain 1) and the RailCab system (domain 2) is provided. In this approach an interface ontology forms the basis of the cross-domain communication. In addition the communication procedure of the agent interaction, as described by the FIPA standard, is outlined in an FIPA conform auction between the production facility and RailCabs within the JIT radius. The JIT radius forms a set of suppliers. They have in common, that they are able to deliver the required goods within the given time. Therefore, the goal of the auction is to identify the lowest priced RailCab. During the auction RailCabs may occur as a team. Negotiation and voting procedures enables each RailCab to submit an optimal offer for the team.

#### 5.3.11.1 T-Box Design

The following two ontologies are designed to demonstrate in which way the T-Box design of underlying ontologies can be done. It is important to point out, that there is a so called *Open World Assumption* given. This implies the need to explicitly rule out unwanted facts within the ontology design. Another aspect of central importance for the successful deployment of ontologies is their integration. Unfortunately, this is not trivial in general. Figures 5.44 and 5.45 show the conceptual approach to be proportionate to the problem, based on [19] and [12]. Figure 5.44 shows that a distinction is done within the facility (domain 1) between the following levels:

- The *Foundational Ontology* includes the abstract concepts of time, space, object, event, etc. As well as the concepts of major priority for this context, as there are *transporter*, *product* and *package*. It is desired that only one *Foundational Ontology* exists within this design, which serves as a starting point for modelling the *production-side* and the *RailCab-Interface-Ontology*. In this manner, the other ontologies can be seen as specializations and by thus allow integration. The *Foundational Ontology* forms a key specification that allows more specialized ontologies to model redundant concepts.
- The *RailCab-Interface-Ontology* provides the communication interface to the RailCab agents.
- The *domain1-ontology* represents the entire ontology structure, that is available for domain 1.

Figure 5.45 provides an overview of the integration approach for the mentioned ontologies in more detail, with respect to RailCab agent system. It should be noted that both, the already known *Foundational Ontology* and the *RailCab Interfaceontology* can be integrated into the *domain2-Ontology* in the same way. The ontology of the RailCab is similar to the one of the production side in a way, that both share the concept of the *Foundational-Ontology* as well as the terms of the *RailCab interface-ontology*.

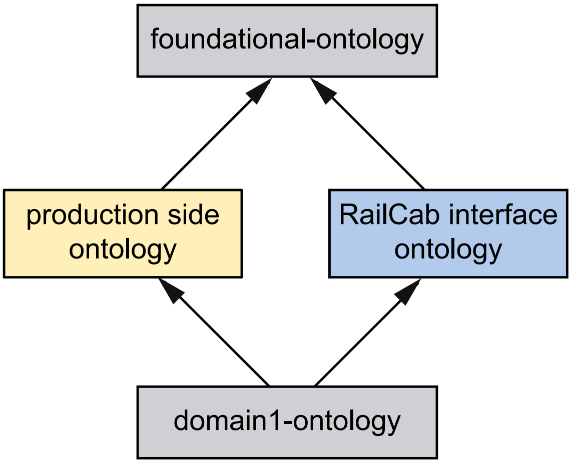


**Fig. 5.44** Three levels of ontology generalization regarding the production facility (domain 1)

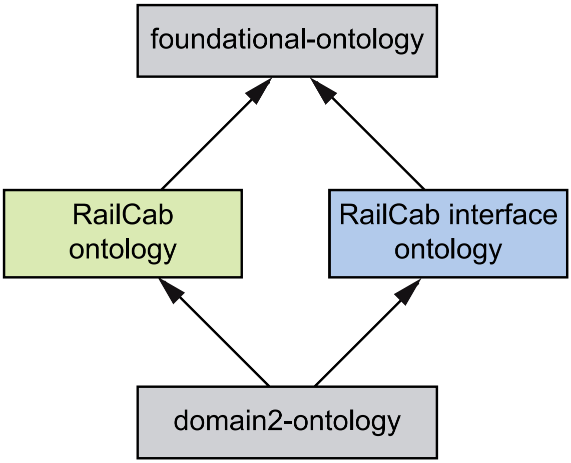


**Fig. 5.45** Three levels of the ontology generalization regarding the RailCab system (domain 2)

#### 5.3.11.2 Communication Flow

In this section, a communication protocol is presented that allows the communication between the production site (domain 1) and the RailCab system (domain 2). The shown communication process meets the specifications by FIPA standard. In this, the production side will request RailCabs to make an offer regarding to the transportation of a specified delivery. The contacted RailCabs may be associated with a supplier's fleet, with teams trying to optimize their company's profit. This is realized by distributing received auctions towards their team members as part of a negotiation. In order to vote for the RailCab which offers the best conditions for the group to take over the job. The figure 5.46 illustrates the underlying connection graph of a single RailCab team. Highlighted are the agents *RailCab A* and *C*, and *productionside* because they are particularly interesting for the further consideration of the communication sequence.

It is necessary to find an efficient voting algorithm which allows the RailCabs to optimize their teams benefit within the given time. The problem is equivalent to the principle *Leader Election* problem where agents are differentiated on the basis of a utility function. Since this may not be clear, however, ambiguity of the function is not relevant for maximizing the supplier's earnings.

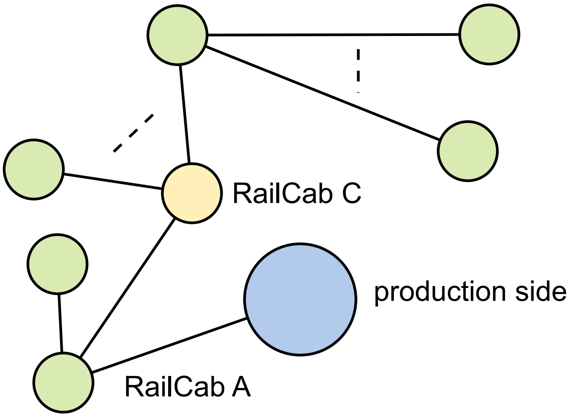


**Fig. 5.46** Arbitrary connection graph of Railcab units and the production site

It must also be assumed that not every member of the team can exactly name all other team members that are relevant for the problem. Since individual data might not be actual, or communication might not be successful within the given deadline by the production side (cf. Fig. 5.46). The voting problem in principle is a *Leader Election* in a spanning tree with asynchronous communication. It is therefore useful to take advantage of the *FloodMax* approach here. Unfortunately, the algorithm is generally in arbitrary graphs with asynchronous communication is very difficult to use (cf. [139]). Therefore the algorithm will be used in an optimized form regarding to the problem.

The voting procedure can be divided into several sub-routines. Figure 5.47 illustrates the reaction of the RailCab receiving a `call-for-proposal` message. It sends an `inform` message to all known team members containing the following:

- The original `cfp` message, that was send from the production site, this contains all the terms of the offer and a deadline until proposals in the form of `proposal` messages have to be done.
- The value that was determined by the utility function, with respect to the auction. This should be the benefit of the team as the transmitter can achieve if it would accept the job.

Each receiver of such an `inform` message has to check weather the utility function may result in an higher value, with respect to his individual parameters. In case of a higher value the receiver knows that a better result for it's team can be achieved by taking the job, rather than the team member, that has send the `inform` message.

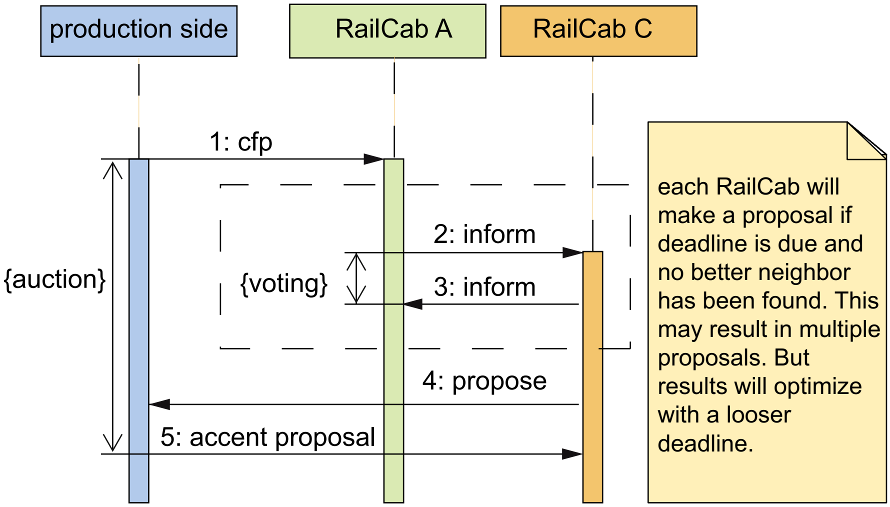

production side
RailCab A
RailCab C
1: cfp
2: inform
3: inform
4: propose
5: accent proposal
{auction}
{voting}
each RailCab will make a proposal if deadline is due and no better neighbor has been found. This may result in multiple proposals. But results will optimize with a looser deadline.

**Fig. 5.47** Sequence diagram communication flow

If the deadline, given by the production side within the initial `cfp` message, is about to expire, each RailCab is in a difficult situation. The potential team benefit can only be optimal if and only if the production side gets proposals from team members, which can grant a maximal profit. Because the production side itself does not differ between the optimal and suboptimal team members, because there is no data nor interest about it, it may choose randomly among the best bids. In order to maximize the potential team profit, it would be best for those suboptimal team members to ignore the auction, by this they higher the probabilities of each optimal team member to gain the job. Figure 5.47 shows an example for the flow of communication, the presentation is limited for reasons of clarity to only three RailCabs of the same team. It can be seen how the auction is initiated and in which way the negotiation of RailCab voting takes place.

### *5.3.12 Preparing Solution Pattern "Hybrid Planning"*

Roman Dumitrescu and Harald Anacker

To enable self-optimization in mechatronic systems, planning methods are of high importance. However, classic planning methods consider state transitions as a black box, so only the state before and after the transition will be accounted for the self-optimization process. In mechatronic systems the continuous run of the processes taking place within the system should not be neglected in order to avoid deviations during the execution of a plan. As a consequence, mechatronic processes have to be described in a continuous way, but also needs to be planned. The **solution pattern** "Hybrid Planning" is based on the detailed method for the behavior planning in non-deterministic environment which was explained before. Core of the solution

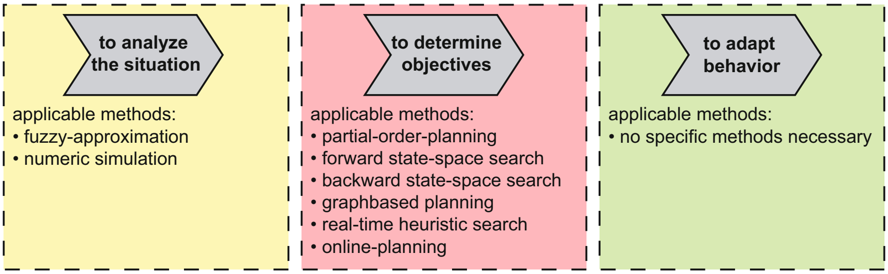


**Fig. 5.48** Possible methods fort he implementation of the solution pattern "Hybrid Planning"

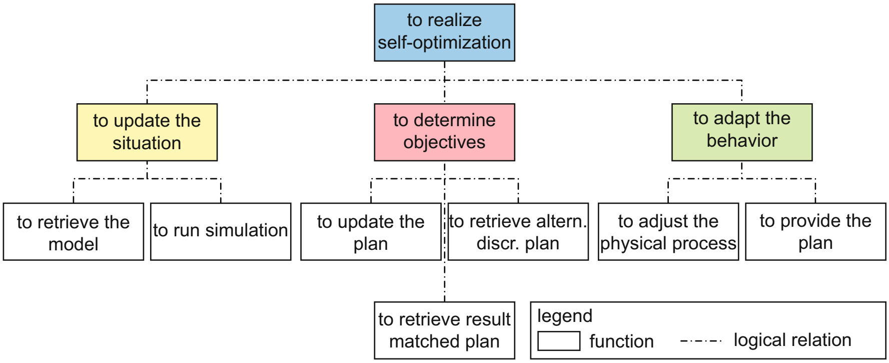


**Fig. 5.49** Partial model behavior–activity of the solution pattern "Hybrid Planning"

pattern is the combination of classical planning algorithms with methods for the approximation of continuous behavior. Regularly the approximation is realized by a simulation model and an update of the existing plan. The solution pattern could be realized by different combination of methods that are illustrated in Fig. 5.48. The different methods are allocated to the different phases of the self-optimization process.

The main planner is subdivided in two different planners. A discreet planner generates the offline plan before the system starts running. Depending on the different usage conditions additional planners are necessary, for example to cooperate with additional (sub)systems. Figure 5.49 shows the partial model functions of the solution pattern "Hybrid Planning". A discrete planning method "determines the objectives" to generate plans or partial plans, whereas the continuous parts of the planning focuses on "to update the situation" for the evaluation of the planning steps. Merging the results of the continuous planning into the discrete planning results in "adapting the behavior" whether by "providing the plan" or by "adjusting the physical process".

**Fig. 5.50** Partial model active structure of the solution pattern "Hybrid Planning"

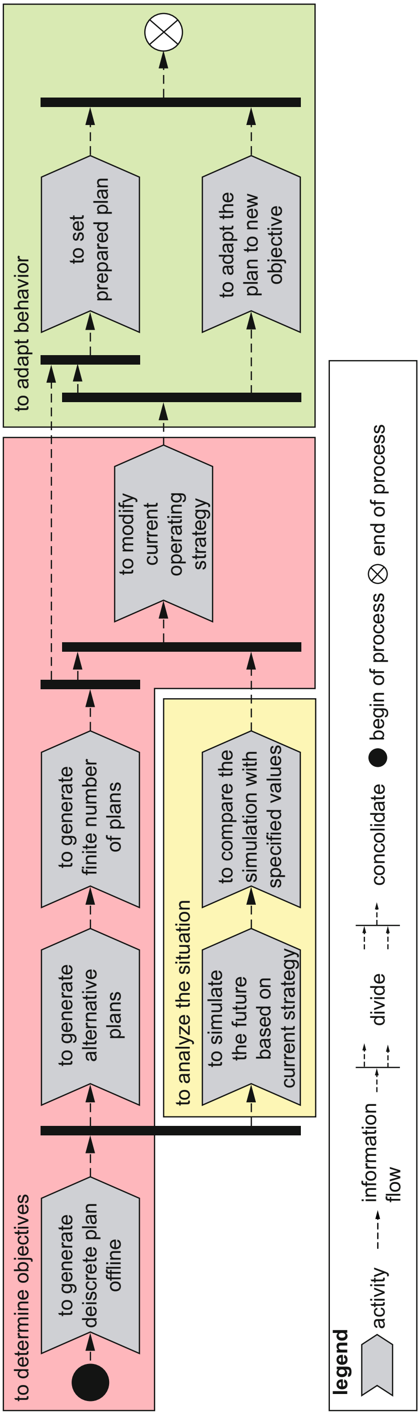
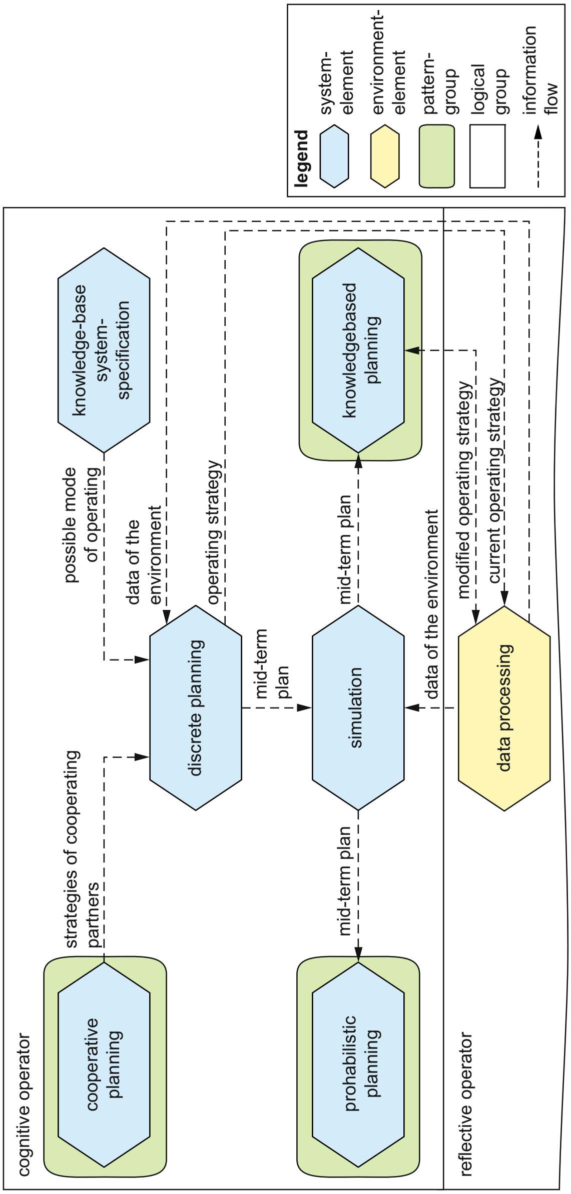

to determine objectives
to generate deiscrete plan offline
to generate alternative plans
to generate finite number of plans
to analyze the situation
to simulate the future based on current strategy
to compare the simulation with specified values
to modify current operating strategy
to adapt behavior
to set prepared plan
to adapt the plan to new objective
legend
activity
information flow
divide
concolidate
begin of process
end of process

**Fig. 5.51** Partial model functions of the solution pattern "Hybrid Planning"



The procedure of the hybrid planning takes place in several steps (cf. Fig. 5.50). As opposed to the three actions of the self-optimizing process, the hybrid planning has at the beginning four main activities, which can be classified into the three steps of self-optimization anyhow. Even before the "Analysis of the situation" takes place, there is an initial offline planning which determines the initial objectives (red element in Fig. 5.50). Then, the plan actions are analyzed in simulation by comparing the plan with the current situation and the current plan gets modified or rescheduled, not only by calculating alternative plans, but also potentially by using the data from the simulation steps. Eventually the currently active plan gets executed.

For the implementation of the functions of the solution pattern "Hybrid Planning" the following essential system elements and arrangement of them were identified (cf. Fig. 5.50).

The system elements "offline planning" and "predictive planning" are realizing the function "to retrieve alternative discrete plan", which is illustrated in Fig. 5.51. The "online planning" holds the function "to retrieve result matched plan". The system element "approximation of continuous behaviors" carries out the functions "to run simulation", "to retrieve the model" and "to run simulation".

## 5.4 Dynamic Reconfiguration

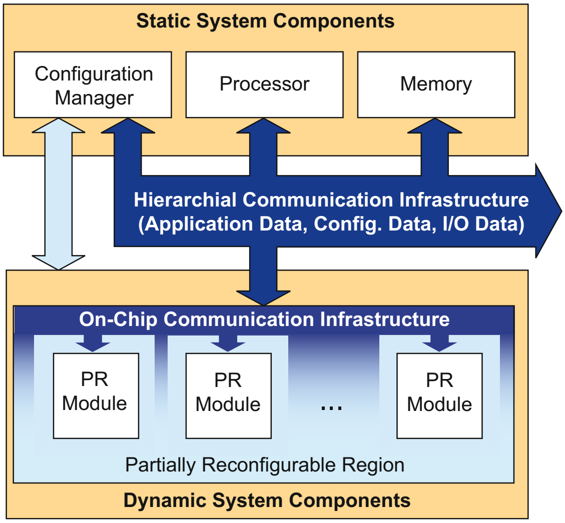
Sebastian Korf and Mario Porrmann

When principles of self-optimization refer to the topology and structure of **microelectronic systems**, a reconfiguration of the system architecture or of the dedicated system components is required. In this context, reconfigurability means the possibility to change the functionality or interconnection of hardware modules in microelectronic systems before and during operation. We distinguish between fine-grained (FPGA-based) and coarse-grained (processor-based) reconfigurable architectures. These architectures assign two different hardware technologies for the process step "Selection of Hardware Technology" in the design and development of electronic engineering in Sect. 3.3.4 on page 88. In Sect. 5.4.1, fine-grained FPGA-based dynamically reconfigurable systems are introduced which facilitate System on Programmable Chip (SoPC) designs with a complexity of several million logic gates, several hundred kBytes of internal SRAM memory, and embedded processor cores. Section 5.4.2 will detail our work on embedded processor cores that can adapt their internal structure at run-time. In Sect. 5.4.3, two modeling approaches for reconfigurable architectures are described, which are used to determine the appropriate model for the process step "Modeling of Information Processing Dynamic Reconfigurable Hardware" in the design and development of electronic engineering. The modeling approaches are used in Sect. 5.4.4 for the design of a dynamically reconfigurable system. The design methods are used within the process steps "Modeling of Information Processing Dynamically Reconfigurable Hardware" to "Synthesis of Dynamically Reconfigurable Hardware". Section 5.4.5 concludes with concrete applications for fine-grained and coarse-grained architectures.

### *5.4.1 Fine-Grained Reconfigurable Architectures*

**FPGA-based reconfigurable systems** try to fill the gap between flexible, programmable microprocessors and application-specific hardware with respect to cost, energy-efficiency, and performance. Partially and dynamically reconfigurable

**Fig. 5.52** Architecture of a dynamically reconfigurable system


Static System Components
Configuration Manager
Processor
Memory
Hierarchial Communication Infrastructure (Application Data, Config. Data, I/O Data)
On-Chip Communication Infrastructure
PR Module
PR Module
...
PR Module
Partially Reconfigurable Region
Dynamic System Components

systems add an additional level of flexibility since the functions and interconnectivity of their hardware resources can be changed during run-time. In this way, the architecture can be flexibly adapted to changing environmental conditions. The traditionally static partitioning into hardware and software can be replaced by a dynamic partitioning at run-time. Therefore, dynamically reconfigurable hardware is a promising technology for information processing in self-optimizing systems. Nevertheless, these methods are rarely used in real-world applications due to a lack of sophisticated design tools that support partial reconfiguration. Therefore, new design methods and new hardware platforms have been developed, which enable an efficient utilization of dynamically reconfigurable systems.

Figure 5.52 shows the system architecture that is used for the implementation of FPGA-based dynamically reconfigurable hardware. The FPGA resources are divided into a static and a partially reconfigurable region (PR region), connected by a hierarchical communication infrastructure. The static region typically comprises of one or more processors, embedded memory, and a configuration manager that manages the available resources, configuration files, and the reconfiguration process. The dynamic system components are represented by partial reconfiguration modules (PR modules) and the placement of a PR module is done by configuring a predefined area in a PR region of the FPGA with the corresponding configuration data. PR modules can be loaded into or erased from the system during run-time. Communication between PR modules as well as with the static components is realized by a flexible on-chip communication infrastructure. Using state-of-the-art FPGAs enables the realization of complete systems on one chip, since these devices provide all the logic resources that are required.

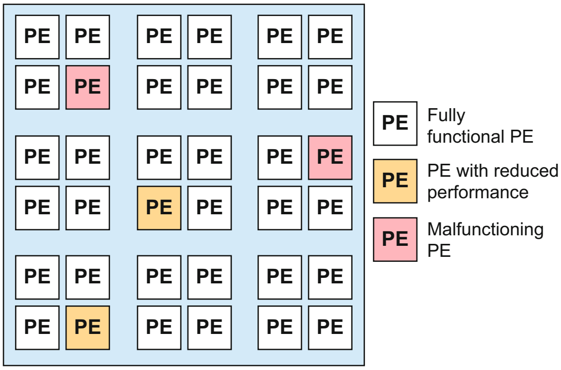


**Fig. 5.53** Multiprocessor with processing elements in different conditions

## *5.4.2 Coarse-Grained Reconfigurable Architectures*

The high flexibility of fine-grained reconfigurable systems, like FPGAs, comes at the cost of high overhead in terms of chip area, timing delays, and power. An alternative to dynamically reconfigurable FPGA-based systems are **Multi Processor System on Chip** (MPSoC) architectures, which are also able to cope with today's requirements on short time-to-market due to manageable design complexity, high energy efficiency in spite of high performance, and high reliability [210]. Here, we target on-chip multiprocessors composed of hundreds of simple embedded processors, connected by a network on-chip (NoC) [107]. In these architectures, the inherent redundancy can be utilized to increase reliability and system lifetime [165].

As illustrated in Fig. 5.53, it is expected that future on-chip multiprocessors will comprise a growing number of processing elements. Some of them will probably be malfunctioning or provide only reduced performance, e.g. due to semiconductor parameter variations. Unfortunately, more and more of these system faults occur dynamically during operation. The goal of our approach for future self-optimizing MPSoCs is to provide the user with the maximum performance of energy efficiency that can be achieved in the actual system state by utilizing as many hardware building blocks of the architecture as possible. Therefore, we integrate methods for dynamic reconfiguration into the architecture, which enable reconfiguration of the interconnection between the building blocks of the processors at run-time. Details about these methods are described in Sect. 5.4.5.3.

## *5.4.3 Modelling*

The realization of dynamically reconfigurable systems requires a complex design flow that cannot be established based on commercially available tools. Therefore, two modelling approaches will be introduced. In Sect. 5.4.3.1 the PALMERA model abstracts the design on different layers. The DMC model described in Sect. 5.4.3.2 further introduces analysis methods and concepts.

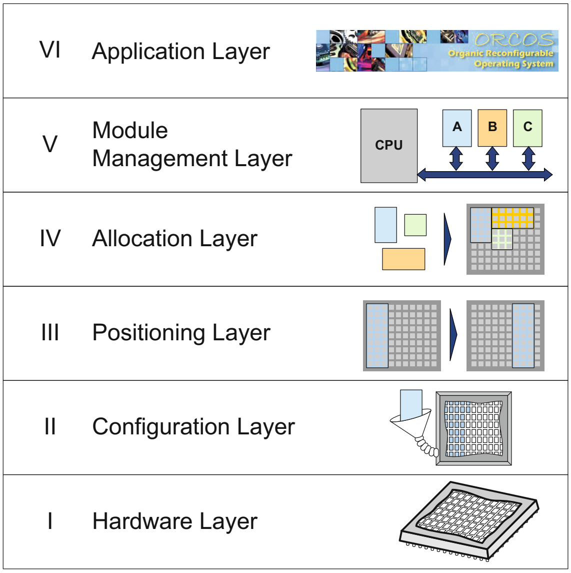


**Fig. 5.54** PALMERA – Paderborn Layer Model for Embedded Reconfigurable Architectures

#### 5.4.3.1 PALMERA (Paderborn Layer Model for Embedded Reconfigurable Architectures)

In order to realize dynamically reconfigurable systems, we propose a layer-based approach to dynamic reconfiguration in [116]. This model systematically abstracts the underlying reconfigurable hardware to the application level by means of six specified layers and well defined interfaces between these layers, as depicted in Fig. 5.54. The main objective is to reduce the error-proneness of the system design while increasing the reusability of existing system components. Additionally, it can be used for the comparison and consolidation of various approaches to dynamic reconfiguration that have been proposed in literature. Each layer offers services to the next higher layer and makes requests of the next lower layer. As for other known layer models in computer science and engineering, the interfaces between layers are standardized to enable an easy and separate exchange of single layers without modifying the whole system.

The first layer in PALMERA is the **Hardware Layer**, representing the underlying reconfigurable hardware. As such it is defined after choosing an FPGA architecture for the system. The interface to its adjacent layer is the configuration port of the chosen FPGA. This makes the interface between the Hardware Layer and the Configuration Layer the only non-specified interface in our model. It is the task of the Configuration Layer to adapt to this interface.

The purpose of the **Configuration Layer** is to abstract from the underlying hardware and its configuration port and to give a standardized interface to the Positioning Layer. For Xilinx FPGAs, the configuration ports are typically either the internal configuration access port (ICAP) or an external configuration port such as the SelectMAP interface. It should support write and read-back of partial bitstreams as well as a complete configuration with a complete bitstream. Due to the streaming-based configuration interfaces of common FPGAs, the Configuration Layer can efficiently be realized in hardware. To shorten configuration time and to avoid storing the bitstreams in the Configuration Layer before configuring, the interface to the Positioning Layer should offer a streaming-based data input for incoming bitstreams as well as a data output for storing the information, which was read back from the FPGA.

The **Positioning Layer** adapts the position information of a given bitstream to a desired location on the FPGA. This can significantly reduce the number of bitstreams that have to be stored for each module since all equivalent (homogenous) areas on the FPGA can be configured with the same bitstream in this case. This also applies to existing heterogeneous architectures if the placement is chosen appropriately. The Positioning Layer thus performs a bitstream manipulation that can be done in software (e.g. with PARBIT [103]), or in hardware (e.g. with REPLICA [112]). However, a hardware implementation of the Positioning Layer is preferred, since it can be realized using only a few resources, without increasing the configuration time significantly. The Positioning Layer has a separate interface to the memory holding the partial bitstreams. It is the uppermost layer that deals with bitstreams as physical representations of the modules. The three upper layers treat the modules as abstract units. Hence, the interface to the Allocation Layer consists only of control flow signals. The services offered to the Allocation Layer are loading and reading configuration data to/from a given area of the FPGA. In addition, combined reading and writing should be offered, in order to shift active modules as needed for a defragmentation of the FPGA.

The **Allocation Layer** manages all available reconfigurable hardware resources on the FPGA and assigns appropriate positions to incoming modules. Therefore, the Allocation Layer holds an abstract image of the resources which can be allocated and deallocated during run-time. In addition, a list of all currently loaded modules is stored in this layer. It holds information about the modules' names, positions, status (active, inactive, etc.), module type and a unique ID. This ID is used to identify a module within the upper two layers. The mapping of a module to an area on the FPGA is done according to a given placement strategy, such as first fit, best fit, or even more sophisticated strategies for heterogeneous FPGAs, as proposed, e.g. in [121]. When needed, the possibility to defragment the FPGA area can also be implemented in the Allocation Layer. A defragmentation can be accomplished automatically (e.g. when a certain degree of fragmentation is reached) or it can be done on-demand. The Allocation Layer offers the service to place a module of a given type on the FPGA or to delete a module with a given ID. The latter is realized by loading an empty bitstream to the FPGA (as required for some fine-grained placement approaches) or by simply deallocating the used resources.

The **Module Management Layer** completely abstracts from the reconfigurable hardware. Its main service offered to the Application Layer is to provide access to a module of a requested type. For this reason it holds a list of all currently loaded modules. With this list a set of different strategies can be implemented, e.g. a caching of unused modules. For this strategy, modules are set to inactive after being released from an application. In case an application needs a module of which an inactive instance exists, a time consuming configuration can be avoided by just reactivating the concerned module. Inactive modules get deleted from the FPGA as soon as the Allocation Layer runs out of free resources. In this case the Module Management Layer chooses a module to be deleted. This can be done according to different strategies such as longest-unused-module-first or module priorities.

The **Application Layer** represents any task using the dynamically reconfigurable hardware. This could be either software running on a (embedded) processor, such as an operating system, or other static hardware modules. In the last-mentioned case it is possible that multiple applications use the dynamically reconfigurable hardware modules simultaneously.

Depending on the architecture of the system, all layers can be implemented either in hardware or in software. On the Application Layer e.g. a small reflex operator can be realized as a pure hardware solution or as a complex software solution running on a CPU with an RTOS such as *ORCOS*. PALMERA has been included in an extension of the OS Monta-Vista-Linux, where the bottom layers (up to the Positioning Layer) are implemented in hardware and the upper layers are software implementations running on a PowerPC processor on a Xilinx FPGA [174].

#### 5.4.3.2 DMC Model for Dynamically Reconfigurable Systems

The DMC (Design, Module, and Component) model [124] is used as a basis for the analysis of the methods and concepts for dynamic reconfiguration. The model divides the placement of a hardware module into three levels of abstraction: Design, Module, and Component. It defines the relations between these levels and is restricted to the fundamental measures that are required for the realization of dynamic reconfiguration. Therefore, methods for placement and scheduling in dynamically reconfigurable systems can be formally described using the DMC model. In the DMC model, reconfigurable architectures are modeled as reconfigurable cells, which are arranged in a matrix structure and interconnected by a communication infrastructure. Fine-grained architectures like FPGAs can be modeled as well as coarse grained and heterogeneous architectures. A design in the DMC model represents an abstract specification of the hardware design, e.g. based on a hardware description language or a schematic. The term module refers to a specific implementation of a design, e.g. generated by a hardware synthesis. Finally, the component represents an instance of a module. Several instances of the same module may be placed in parallel at different positions on the reconfigurable hardware. For the analysis of architectures and methods based on the DMC model, we have developed the simulation framework SARA (Simulation Framework for Analyzing Reconfigurable Architectures). SARA is specifically designed for FPGA-based architectures. The simulation flow of SARA is split into three phases. In phase one, a Virtual Synthesis tool creates the modules to be downloaded to the FPGA from a given set of module descriptions. These descriptions include information about the required FPGA resources as well as minimum module dimensions. According to one or more given synthesis strategies, module implementations with various aspect ratios are generated for each module description. In phase two, an RTR-manager (run-time reconfiguration manager) executes the given benchmark and places the required modules in a predefined order onto a virtual FPGA. The simulation analysis is done in phase three by a dedicated analysis tool integrated in SARA. In the context of self-optimizing systems, SARA is specifically used for the analysis of new placement and defragmentation strategies [120, 122, 123].

## *5.4.4 Design Methods for Dynamic Reconfigurable Systems*

Based on the abstract modelling of PALMERA and the DMC model, the Integrated Design Flow for Reconfigurable Architectures (INDRA) has been developed that guides the designer through the different implementation steps to create a concrete dynamic reconfigurable system architecture. This design flow is described in Sect. 5.4.4.1. Section 5.4.4.2 introduces algorithms for the flexible placement of dynamically reconfigurable (hardware) modules. A design method for a Hardware-in-the-Loop (HiL) implementation is shown in Sect. 5.4.4.3.

### 5.4.4.1 INDRA (Integrated Design Flow for Reconfigurable Architectures)

The design-flow of a partially reconfigurable system is different from the standard design-flow of reconfigurable systems, which only allows the reconfiguration of the whole FPGA. To efficiently handle these deviations from the standard flow, the Integrated Design Flow for Reconfigurable Architectures (INDRA) has been developed (cf. Fig. 5.55). INDRA integrates all tools that are required to design dynamically reconfigurable systems based on Xilinx FPGAs [88]. It combines commercial state-of-the-art tools and tools that have been adapted or especially designed for this framework. INDRA supports a flexible one-dimensional or two-dimensional module placement.

First, the given application is partitioned into static and dynamic system components. The area that is used for the static components is also referred to as the base region. The partitioning depends on the properties of the selected device, such as the reconfiguration granularity (length of a so-called configuration frame), and on the selected placement approach. The architecture of Xilinx Virtex-4 to Virtex-7 devices allows a two-dimensional placement of partial reconfiguration modules at a granularity of a configuration frame. The description of the static and dynamic system components as well as their interconnections between each other on the top level are specified in a hardware description language (HDL). The synthesis of the base region and of the PR Modules is performed based on the system partitioning. Depending on the size of the modules, which is obtained from synthesis estimation, and on the inherent heterogeneity of the FPGA, INDRA determines the steps required for the synthesis of the PR Modules. The floorplanning of the system (the mapping of each component to a position on the FPGA) is done by SARA, which implements the DMC model for the dynamically reconfigurable system.

In addition to the partitioning and floorplaning of the FPGA, the concept of partial reconfiguration requires a suitable communication infrastructure for

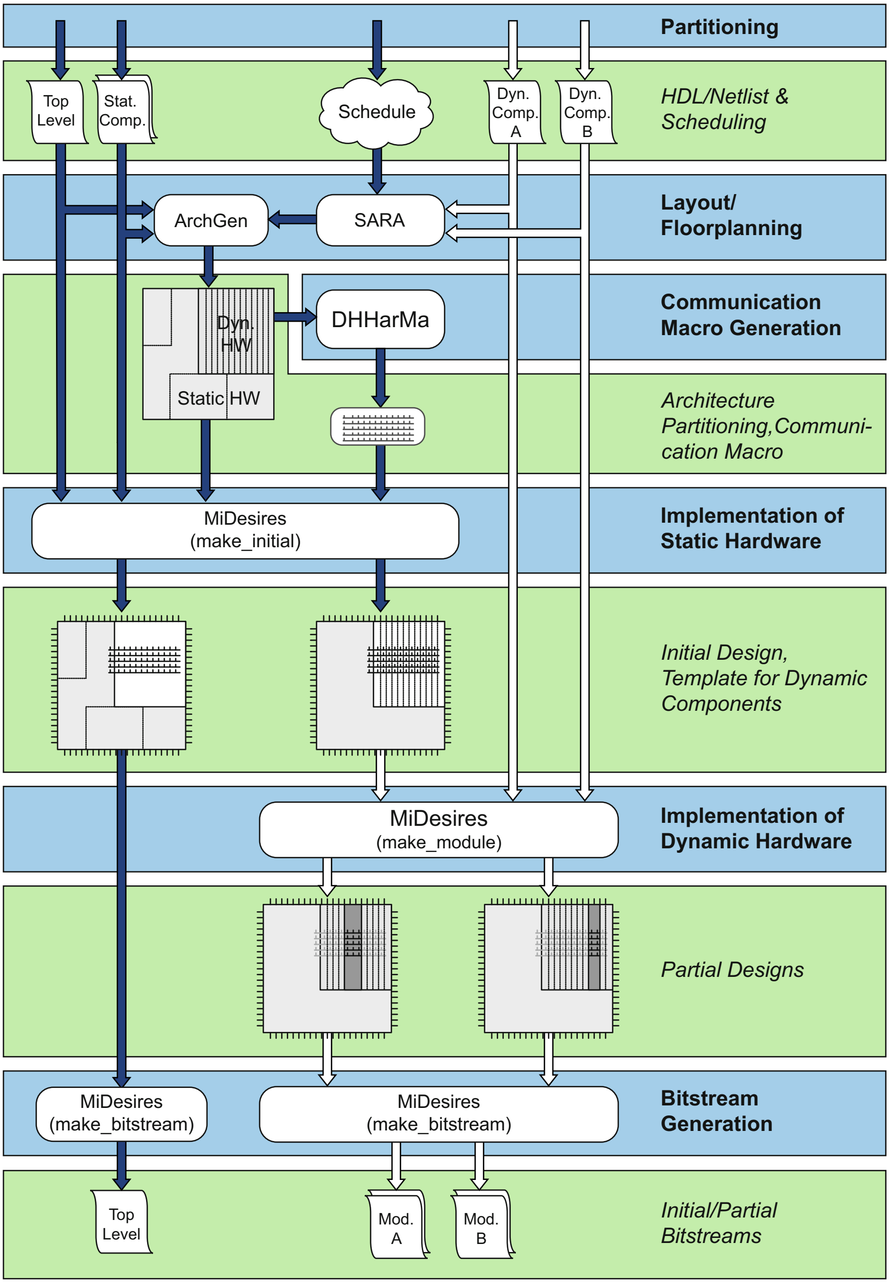

Partitioning
Top Level
Stat. Comp.
Schedule
Dyn. Comp. A
Dyn. Comp. B
HDL/Netlist & Scheduling
ArchGen
SARA
Layout/ Floorplanning
Dyn. HW
Static HW
DHHarMa
Communication Macro Generation
Architecture Partitioning, Communication Macro
MiDesires (make_initial)
Implementation of Static Hardware
Initial Design, Template for Dynamic Components
MiDesires (make_module)
Implementation of Dynamic Hardware
Partial Designs
MiDesires (make_bitstream)
MiDesires (make_bitstream)
Bitstream Generation
Top Level
Mod. A
Mod. B
Initial/Partial Bitstreams

**Fig. 5.55** INDRA – Integrated Design Flow for Reconfigurable Architectures

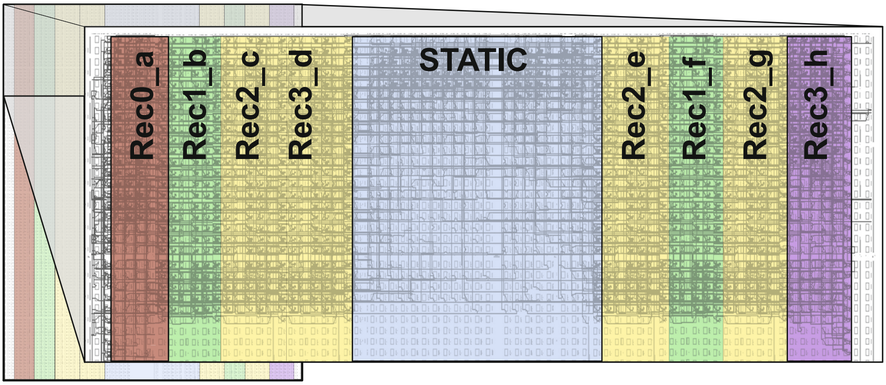


**Fig. 5.56** Example of a homogeneous hard macro for a communication infrastructure with 9 regions and 4 different types of regions

interconnecting the PR modules and the base region. The communication infrastructure should not introduce any further heterogeneity in the system to maintain the flexibility of placement by preserving the number of feasible positions of the PR modules. Homogeneity implies that the individually reconfigurable tiles (a tile is the atomic partially reconfigurable unit, cf. 5.4.4.2) are connected by the same routing resources. Thus, modules cannot only be placed at one dedicated position, but at any position with sufficient free contiguous resources [87]. Current commercially available FPGA place and route tools lack an option for generating this type of homogeneous designs. The Design Flow for Homogeneous Hard Macros (DHHarMa) [126] targets the automatic generation of homogeneous and regular designs starting from a high-level description, such as VHDL or Verilog. Using DHHarMA, complex communication infrastructures for dynamically reconfigurable systems can be generated based on an abstract high-level description. In [126], examples are presented, using 32 Bit data, 32 Bit addresses, 4 Byte-enable signals, and 4 Bit auxiliary lines. Additionally, dedicated signals are connected to each region for strobe, master request, master grant, region enable, and region reset. The communication infrastructure also supports bursts (transmission of multiple data packets at a time) using an embedded 8 Bit burst counter (cf. Fig. 5.56).

Figure 5.57 shows an example partitioning of a Virtex-4 FX100 FPGA. In the Virtex-4 architecture, the area of a PR region should be multiples of a configuration frame. In the example implementation, we vertically divided the FPGA, so that the resources located left of the center column are dedicated to static system components, and the resources located right of the center column are considered for the tiled PR region.

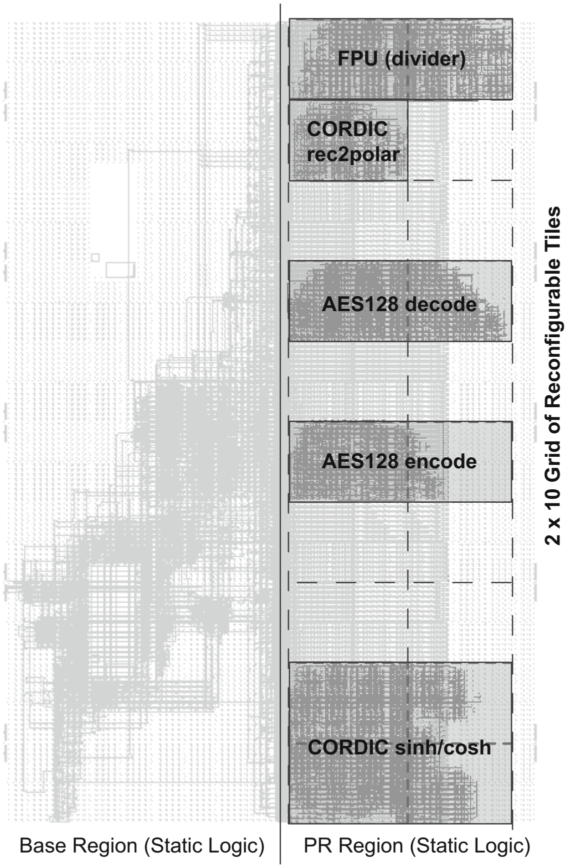


**Fig. 5.57** Example for the partitioning of a Xilinx Virtex-4 FX100 FPGA

### 5.4.4.2 Placement Algorithms for Flexible Dynamically Reconfigurable Systems

Nowadays, most realizations of dynamically reconfigurable systems use simple approaches that are based on fixed module slots. The placement flexibility of these implementations is different from the flexibility assumed and analyzed in the theoretical research work. In [123] we present ways to help close this gap by showing how today's heterogeneous FPGAs can be used for dynamic reconfiguration with free module placement, varying module sizes, and multiple instances of modules.

In a tiled partially reconfigurable system as described in [87] the partially reconfigurable region is subdivided into reconfigurable *tiles*. Tiled partitioning allows for the placement of multiple PR modules with various sizes in a PR region. A reconfigurable tile can be considered as an atomic unit of partial reconfiguration. A PR region may contain several different types of tiles offering different amounts of available resources. The tile sizes may vary according to the different resource types within each tile.

**Fig. 5.58** Example of a partitioning scheme using a PR region with reconfigurable tiles

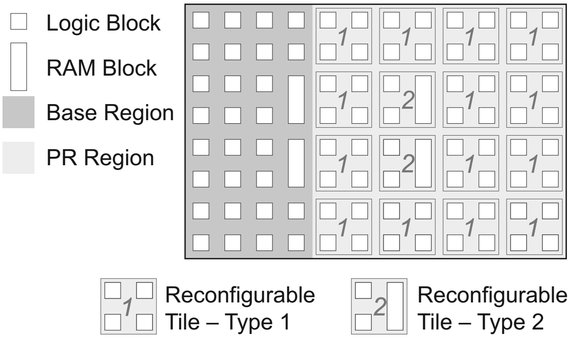


**Fig. 5.59** Example of a set of PR modules and their feasible positions

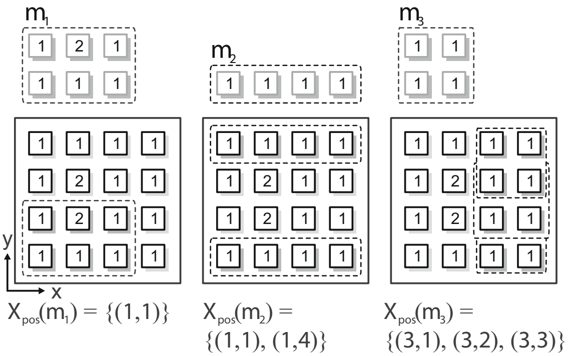


Figure 5.58 shows an example with a base region and a PR region, which is partitioned into an area of 4 × 4 reconfigurable tiles. The PR region in the example is heterogeneous, since two different types of tiles are used. At run-time, an instance of a PR module is mapped to one or several contiguously aligned tiles. This is done by partially reconfiguring the selected tiles using the equivalent configuration data (partial bitstream) of the PR module. A PR module can occupy any size from a single tile to all tiles of the PR region. Figure 5.59 shows the PR region of Fig. 5.58 and an example of a set of PR modules with the corresponding feasible positions. The values in each tile indicate the type of the tile.

With respect to run-time placement, the PR modules vary according to their resource requirements, their shape, and their feasible positions. Each feasible position of a PR module can have a different degree of overlap with the feasible positions of the other PR modules in the system. The degree of overlap has an impact on the placeability of the PR module. Those feasible positions that overlap with many other feasible positions are likely to be blocked by a previously placed instance of another PR module. Thus a reasonable online placement policy is to always select the free position with the least degree of overlap as discussed in [121]. Besides maintaining a large number of free positions at run-time, it is also possible to optimize the placeability of PR modules at design-time. This is done by minimizing the degree of overlap of the feasible positions of the given PR modules. At design-time, the set

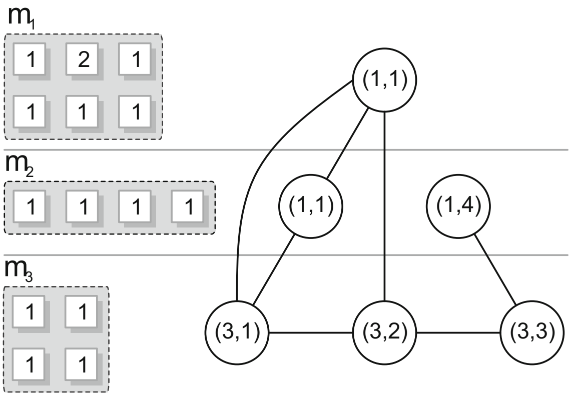


**Fig. 5.60** Example of an overlap graph

of feasible positions of a PR module is defined by the shape and position of the synthesis region. The optimization of the placeability is done by selecting the synthesis regions of the PR modules that allow the best possible placement at run-time.

In order to optimize the placeability of the PR modules, a metric is required, which quantifies the degree of overlap of the feasible positions. The overlap graph $G = (V,E)$ is an undirected graph, where V are the nodes and E the edges between the nodes, that enables visualizing these resource dependencies. It shows which of the feasible positions of the PR modules overlap with each other. The graph can be used with arbitrarily shaped PR modules. For simplicity we will focus on rectangular PR modules. A vertex $v = (m,x,y) \in V$ represents a feasible position $(x,y) \in X_{pos}(m)$ of the PR module $m \in M$. The set of all vertices is defined as

$$V = \bigcup_{m \in M} \{(m,x,y) \mid (x,y) \in X_{pos}(m)\}. \tag{5.22}$$

Hence, the number of vertices is the same as the sum of feasible positions of all PR modules. For a vertex $v_1 = (m_1,x_1,y_1) \in V$ and a vertex $v_2 = (m_2,x_2,y_2) \in V$ an edge $(v_1,v_2)$ is created, if $v_1 \neq v_2$ and the area of PR module $m_1$ at position $(x_1,y_1)$ overlaps with the area of PR module $m_2$ at position $(x_2,y_2)$. Figure 5.60 shows the overlap graph for the PR modules of the example in Fig. 5.59.

With the overlap graph, we can evaluate the degree of overlap for each feasible position of the PR modules. For this purpose we introduce the *position weight*. Using the overlap graph, the computation of the position weights is done in two steps. First, the *probability weights*

$$w_p(v) = p_{alloc}(m)/|X_{pos}(m)| \tag{5.23}$$

are computed for each vertex $v = (m,x,y) \in V$, where $p_{alloc}(m)$ denotes the probability of an allocation of the PR module $m$. The probability weight $w_p(v)$ indicates the probability of a feasible position to be chosen, if all tiles in the PR region are available and a random placement is applied.

Secondly, the position weight $w_{pos}(v)$ of a feasible position is computed by summing the probability weights of the adjacent vertices. The set of adjacent vertices $V_{adj}$ is defined as

$$V_{adj}(v) = \{v_{adj} \mid (v, v_{adj}) \in E\}, \tag{5.24}$$

and the resulting position weight is calculated by

$$w_{pos}(v) = w_p(v) + \sum_{v_{adj} \in V_{adj}(v)} w_p(v_{adj}). \tag{5.25}$$

The position weights reflect the degree of overlap. For example, the placement of an instance of $m_2$ at position $(1,4)$ only blocks the position $(3,3)$ of $m_3$, while the placement of an instance of $m_2$ at position $(1,1)$ blocks the positions $(1,1)$ of $m_1$ and $(3,1)$ of $m_3$. Therefore, the position weight $5/18$ of position $(1,4)$ of $m_2$ is lower than the one from position $(1,1)$.

Apart from the design-time aspects, the position weight can also be used for the placement of PR modules at run-time. The placement is done by selecting the available position with the least position weight. This ensures maintaining a large number of available positions for future placements.

A metric to evaluate the degree of overlap of all feasible positions is to generate a weighted sum of the position weights of all feasible positions. As the probability weight $w_p(v)$ reflects the probability of a feasible position to be selected when randomly placing a module, the overlap weight of all PR modules is defined as follows:

$$w_{ovr}(V) = \frac{1}{|V|} \sum_{v \in V} w_{pos}(v) \cdot w_p(v) \tag{5.26}$$

The weighted mean of the position weights is divided by the total number of feasible positions $|V|$ to balance the degree of overlap and the number of feasible positions. The synthesis regions of the given PR modules can be selected in such a way as to minimize $w_{ovr}(V)$. A small $w_{ovr}(V)$ indicates that the overlaps of feasible positions of the PR modules are small. Minimizing the overlap weight aims at maximizing the number of available positions after placement of a PR module at run-time. Thus the overlap weight is a metric for the placeability of all PR modules.

#### 5.4.4.3 Hardware-in-the-Loop

Hardware-in-the-Loop (HiL) simulations are applied in many areas of embedded systems design, but originate from control design, this is still the main area of interest. In [162], three concepts and tools are presented which allow to interface a simulation of the controller's environment (plant) to an actual implementation of the controller on an FPGA. While this usually requires a model of the plant which can be calculated in real-time, we slow the implemented controller down by exploiting special features of digital hardware. In fact, the simulation environment running the plant model gets in charge of the clock of the hardware design. With this technique we can integrate nearly any FPGA based DUT (Design Under Test) into a simulation environment like MATLAB (Simulink), CamelView [148], or ModelSim. This

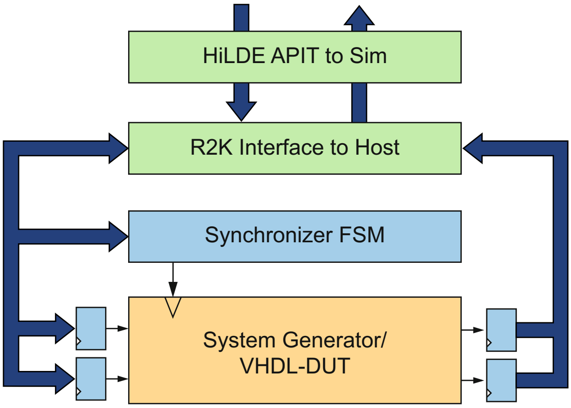


**Fig. 5.61** Non-real-time hardware interface for a Hardware-in-the-Loop system

offline HiL tool flow, called HiLDE (Hardware-in-the-Loop Design Environment) allows for a functional verification of the implemented controller in real hardware, while former test benches from pure software simulations can be reused. Additionally, HiLDE can speed up simulations by several orders of magnitude, depending on the number of in- and outputs and the complexity of the user design. As soon as the DUT is embedded in its target environment, HiLDE cannot be used for testing anymore, as real-time processing is required then. For this, we developed HiLDEGART, (HiLDE for Generic Active Real Time Testing), a tool to visualize and parameterize an active controller in its real environment. Both branches of our tool flow use vMAGIC, an API for the generation and manipulation of VHDL code, to generate the required hardware interfaces as well as configuration data. In the following paragraph, the developed tools HiLDE, HiLDEGART, and vMAGIC are discussed in more detail.

**HiLDE:** The basic idea of our framework is the automatic integration of a DUT into a standardized hardware interface (cf. Fig. 5.61), which enables communication between a simulator and the DUT. This interface consists of a clock controller (Synchronizer) and a set of registers at the inputs and outputs of the DUT. The Synchronizer allows clock cycle accurate control over the DUTs clock by a software environment like Simulink. The input and output registers are used to transfer data between the DUT and the simulation. During a simulation, the three steps 1) write inputs, 2) do n-clock cycles, and 3) read outputs are repeated in a loop controlled by the simulator. The data transfer from the simulation to DUT is done with the Rapid Prototyping Platfrom RAPTOR (e.g. the R2K, cf. Sect. 5.4.5.1). In general, the clock speed in HiLDE simulations will be much slower than the desired clock Speed of the DUT (non-real-time) because the computation of a simulation step of the test-bench or plant-model in the simulator typically takes a lot of time. However, the overall simulation can become faster by several orders of magnitude, if a DUT is moved from the simulation towards hardware.

**HiLDEGART:** After a design has been successfully tested in the HiLDE environment, it can be integrated into its target environment, where it works in real-time.

Still, it is desirable to monitor the controller's IOs as well as its internal states for further testing under real-life conditions. To meet these requirements, HiLDEGART generates hardware interfaces capable of recording signals in real-time without additional external hardware such as logic analyzers. The idea is basically the same as for HiLDE: a hardware wrapper is generated which adds memories to the signals in question and connects those to a bus interface. This enables a GUI to access and display the recorded values. As the output data rates can be much higher than the available communication bandwidth between hardware and software, resampling units and FIFOs are used instead of registers. In addition to that, registers are connected to inputs which parameterize the DUT (such as constants of a controller), so that users can change those values from within the HiLDEGART GUI. Furthermore, the GUI offers advanced features like a triggering unit, which casts events based on boolean operations on the IO signals. This facilitates for example, to increase the resampling rate once a signal reaches a critical level.

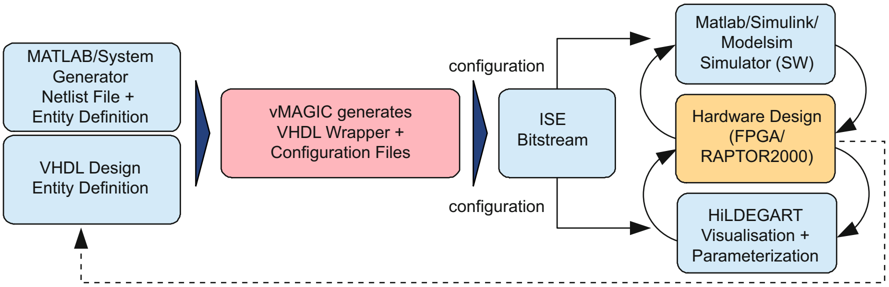

MATLAB/System Generator Netlist File + Entity Definition
VHDL Design Entity Definition
vMAGIC generates VHDL Wrapper + Configuration Files
configuration
ISE Bitstream
configuration
Matlab/Simulink/ Modelsim Simulator (SW)
Hardware Design (FPGA/ RAPTOR2000)
HiLDEGART Visualisation + Parameterization

**Fig. 5.62** Design Flow for HiLDE and HiLDEGART using vMAGIC

**vMAGIC:** The interfaces described in the previous sections are DUT specific and have to be adapted to each new design, which is a tedious and error prone task. As the basic structure stays the same between all implementations, an API was developed, which enables users to write scripts that automatically generate interfaces like these, or automate any other recurring task based on VHDL code as depicted in Fig. 5.62. In [163], the implementation details of vMAGIC are presented.

### 5.4.5 Platforms and Applications

Two main platforms that support dynamic reconfiguration are used in this Section: RAPTOR and BeBot. In the process of developing microelectronic systems, a fast and reliable methodology for the realization of new architectural concepts is of vital importance. Prototypical implementations help to convert new ideas into products quickly and efficiently. Furthermore, they allow for the parallel development of hardware and software for a given application, thus shortening time to market. FPGA-based hardware emulation can be used for functional verification of new MPSoC architectures as well as for HW/SW co-verification and for design-space exploration. The rapid prototyping systems of the RAPTOR family [164] provide the user with a complete hardware and software infrastructure for ASIC and MPSoC prototyping (cf. Sect. 5.4.5.3). A distinct feature of the RAPTOR systems is that the platform can be easily scaled from the emulation of small embedded systems to the emulation of large MPSoCs with hundreds of processors. Along with rapid prototyping, the system can be used to accelerate computationally intensive applications and to perform partial dynamic reconfiguration of Xilinx FPGAs, as presented in Sect. 5.4.5.1.

The BeBot miniature robot, which will be discussed in detail in Sect. 2.2, integrates an embedded processor and a dynamically reconfigurable FPGA (Xilinx Spartan-3). An example for a vision processing application utilizing this architecture will be discussed in Sect. 5.4.5.2.

#### 5.4.5.1 Dynamic Reconfiguration of FPGAs on the Rapid Prototyping Platfrom RAPTOR

The RAPTOR systems follow a modular approach, consisting of a base system and up to six daughterboards. The base system comprises the communication and management infrastructure, used by the daughterboards, which realize the required application-specific functionalities. Because of the modular design, the user can easily integrate new FPGA technologies or communication facilities by means of additional daughterboards. The RAPTOR base system can be integrated into a host PC or run as a stand-alone system. The optional host system can be used to ease monitoring and debugging. For communication with the host system, the RAPTOR-X64 base system integrates a PCI-X and a USB-2.0 interface. The board can be operated outside the normal PCI environment by utilizing the USB-2.0 interface. It is also possible to integrate a PCI-Express-based host-connection by replacing the RAPTOR-X64 by the RAPTOR-XPress baseboard.

The Local Bus and the Broadcast Bus, which are provided with the RAPTOR base system, offer powerful communication infrastructures and guarantee a high-speed communication with the host system and between individual modules. Additionally, direct links between neighboring modules can be used to exchange data with high bandwidth and low latency. Furthermore, all FPGA modules provide additional high-speed serial links for communication between the modules. Reconfiguration (including dynamic reconfiguration) is performed with the maximum possible bandwidth that the FPGAs support. The RAPTOR systems provide a direct migration path from FPGA-based prototypes to ASIC realizations by simply replacing the FPGA based daughterboards with daughterboards that integrate the developed ASICs. Daughterboards integrating different MPSoCs (discussed in detail in Sect. 5.4.5.3) have been realized and can be integrated together with additional dynamically reconfigurable FPGA modules. In this way, information processing systems can be realized that combine the advantages of dynamically reconfigurable FPGAs and MPSoCs.

Figure 5.63 gives an overview of an architecture that has been developed for the realization of self-optimizing drive controllers for a permanent magnet servo

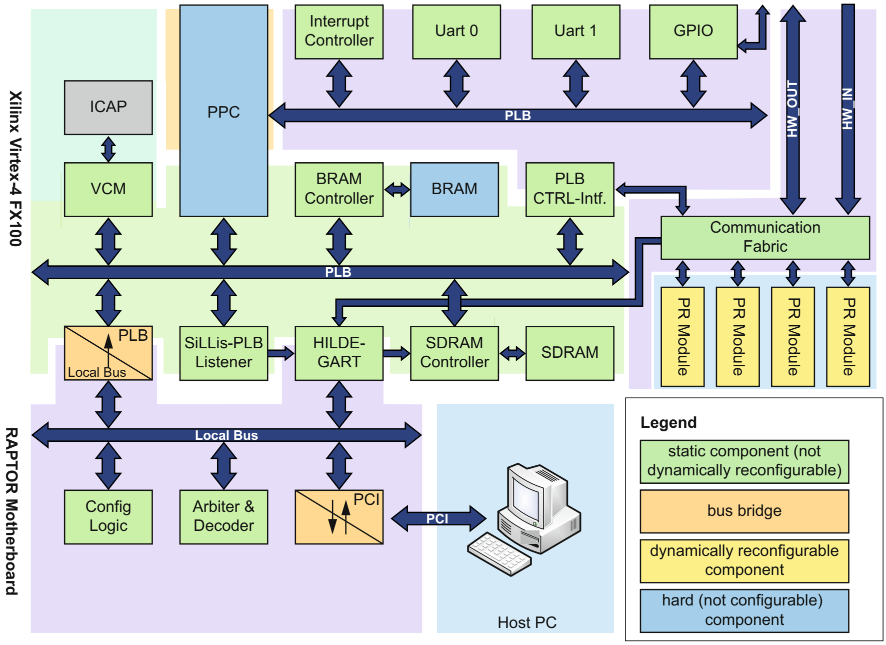


**Fig. 5.63** System architecture for the implementation of a self-optimizing drive controller based on the RAPTOR prototyping system

motor [159]. The implementation is based on the methods for dynamic reconfiguration that have been previously described and is realized on a Xilinx Virtex-4 FX100 FPGA. The architecture is composed of an embedded PowerPC processor connected to dynamically reconfigurable resources (*PR Module*). A processor local bus (PLB) enables communication to the local bus of the RAPTOR system, and from there to the host PC. The dynamically reconfigurable PR modules are used to implement controllers or signal conditioning blocks, since these elements are exchanged according to the current state of the plant and the current objective of the system. The reconfiguration is performed by the Virtex Configuration Manager (VCM) [88].

A program running on the PowerPC initiates the reconfiguration based on a continuous evaluation of the control quality and realization effort, indicating the memory space from the external SDRAM where the partial bitstream ought to be copied. The partial bitstream contains only the needed configuration for one PR module. When a reconfiguration is requested, the VCM initiates DMA transfers from the SDRAM controller, loads the requested partial bitstream to the target PR module by accessing the Internal Configuration Access Port (ICAP), and sends an interrupt to the PowerPC when done. The reconfiguration process lasts about 4.38ms, which represents several control cycles. To overcome this, an initialization routing is used to calculate the initial states of the new-loaded controller. A supervising program, running in the PowerPC, is in charge of monitoring system activity and triggering the dynamic reconfiguration. For the verification of the implemented control algorithms and for testing the correct behavior of the system during dynamic reconfiguration between different controller implementations, we have used the HiL environments HiLDE and HiLDEGART as described in Sect. 5.4.4.3.

#### 5.4.5.2 Dynamic Reconfiguration of FPGAs on the Miniature Robot BeBot

The hybrid processing architecture of the BeBot miniature robot consists of a main processor complemented by an FPGA device that offers on-demand parallel processing with the major advantage that the FPGA can be dynamically reconfigured during runtime to optimally utilize the hardware resources and the energy budget. In contrast to other reported approaches on dynamic hardware reconfiguration, for example [23, 38, 145], we focus on a concept that automatically and dynamically allocates hardware resources depending on the current status of the robot, the required tasks, and the context of operation [149]. Processes can be executed in software on the processor or as modules on the FPGA using partial dynamic reconfiguration. The reconfiguration process is managed by the robot's operating system. The access to the hardware is, from the application point of view, transparent.

Utilizing dynamically reconfigurable hardware for image processing instead of a pure software solution enables real-time image processing and significantly reduces the required computing power of the CPU. Instead, the CPU can be used, e.g. for sensor fusion tasks, behavior generation, and communication within the wireless network. Depending on the current context, hardware configurations can be automatically loaded into the FPGA device, that is, one or more hardware modules are able to process images in parallel. If a specific processing task has been finished, a new hardware configuration can be loaded to optimize the resource utilization on-the-fly.

An initial configuration of the FPGA is loaded after booting the robot. Typically, the local flash memory of the robot is used to store the configuration data. But it is also possible to load the configuration via one of the available wireless communication links. This feature is very useful in multi-robot applications in order to share available processing resources of the robot team by wirelessly transmitting FPGA configuration data to robots that are able to offer computing resources to other robots. This type of resource sharing between wirelessly connected robots, requires additional operating system services, as discussed, e.g. in [80]. In the context of image processing, this can significantly reduce computation time and increase the throughput of images. Reconfiguration of the complete FPGA, requires loading a bitstream file of 728 kByte. On the BeBot, a complete FPGA reconfiguration is performed in 25 ms, corresponding to a reconfiguration rate of 30 MByte/s. If slow communication interfaces are used to transfer the reconfiguration data to the robot, the configuration files can be cached in the internal SDRAM.

Utilizing the INDRA design flow (cf. Sect. 5.4.4.1), partial dynamic reconfiguration of the FPGA can be used to reduce the reconfiguration time. Furthermore, with this concept, it is possible to keep parts of the application and the application data inside the FPGA, essentially reducing communication time. Here, the FPGA is divided into two parts: a static region utilizing 20% of the FPGA resources (slices)

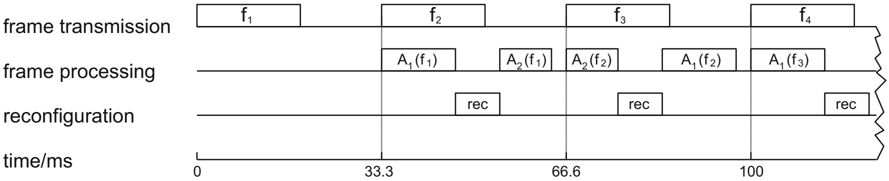


**Fig. 5.64** Dynamic reconfiguration on a frame-by-frame basis

and a dynamically reconfigurable region, comprising 80% of the available FPGA slices. During dynamic reconfiguration, the base region remains unchanged while the partially reconfigurable region is completely reconfigured. Here, partial reconfiguration can be performed in 20 ms if the bitstream is available locally on the BeBot.

Depending on the trigger for dynamic reconfiguration, a differentiation can be made between time-driven reconfiguration and event-driven reconfiguration. In time-driven reconfiguration, the time and order in which PR modules are loaded is known at design-time and do not change at run-time. The reconfiguration controller can be a simple state machine that triggers the reconfiguration at the predefined time intervals. On the one hand, the time between two reconfigurations can be orders of magnitude higher than the reconfiguration time, if complex applications are executed in turn. An example would be changing between two video processing algorithms every 10 seconds. On the other hand, fast partial reconfiguration enables hardware changes at high frequency: in video processing, time-driven reconfiguration can be triggered on a frame-by-frame basis. Figure 5.64 gives an example, where two applications ($A_1$ and $A_2$) are processing consecutive data frames ($f_1, f_2, ...$). It has to be assured that the sum of reconfiguration time and application execution times are lower than 33 ms for the 30 frame per second on BeBot, i.e. 13 ms are available for application execution for the used partitioning, which requires 20 ms for reconfiguration. In the example, the applications are decoupled from the data transmission from the camera since all executions are performed at the previous frame. If this is not possible, a more complex scheduling is required, and application execution typically starts in parallel to data transmission to increase performance.

In the event-driven scheme, the reconfiguration time and the order of the PR modules are not known at design-time. A trigger for dynamic reconfiguration can occur at any time. Changes of the ambient light could, e.g. be used to reconfigure between different video processing algorithms. While the tool flow and the hardware infrastructure are identical for event-driven and time-driven reconfiguration, the implementation of the reconfiguration controller varies. For time-driven reconfiguration, the reconfiguration controller can be realized by a simple timer. In event-driven reconfiguration various internal and external parameters may have to be taken into account to decide when and which PR module to load. On the BeBot the trigger for reconfiguration is set by a software implementation on the internal processor. In both schemes, time-driven and event-driven reconfiguration, the calculation times of this software part of the reconfiguration controller are negligible compared to the FPGA reconfiguration time.

Two different hardware modules have been developed in the context of vision processing. The first provides optical flow motion detection and the second supports color recognition [4, 35, 52]. The optical flow calculation is used to detect walls or obstacles in the operational area and to dynamically construct a map of the environment. The color detection algorithm enhances options to identify objects like marked landmarks or other robots, in order to improve navigation and to map buildings. With the dynamic approach, both algorithms can be processed on the robot platform under real-time constraints achieving a good utilization of the processing devices.

To evaluate the performance of the hardware implementation of these two algrotihms, the BeBot miniature robot prototype has been evaluated in a test room with artificial lighting and a convenient environment. A frame size of 160x120 was chosen, requiring 38,400 Bytes to be transferred from the camera to the SDRAM. The frame rate is fixed to 30 frames per second by the camera used. Therefore, the bandwidth required to transfer the image data into memory is less than 1.2 Mbyte/s. Since the SDRAM can be accessed with more than 80 MByte/s and the FPGA implementation achieves about 46 MByte/s, sufficient bandwidth is available in the system to transfer data between the system components. Performance is mainly limited by the processing time on the FPGA. Wherever possible, communication and calculation are performed in parallel, i.e. calculation starts directly after receiving the first data from the camera, or one frame is processed while the next frame is loaded.

The optical flow does not need any particular parameter updates when the environment is changing, except for the number of columns and the speed threshold. These parameters do not affect the performance nor the area usage of the FPGA-based implementation. A single module implementation of the optical flow requires 1338 slices (18% of the resources available for the partially reconfigurable module) and one frame is processed in 0.9 ms. Processing time includes the time for reading the image data from the SDRAM and for writing the results to the processor.

In contrast, the required FPGA resources and the computation time for the color recognition module can vary significantly depending on the number of maximum recognizable blocks and on the number of colors. In the following paragraph, an analysis of the different configurations on BeBot will be presented, focusing on area consumption and computation time. For the evaluation of the block recognition module, various configurations have been tested. Table 5.4 shows the configurations chosen for a laboratory test. The resource requirements of the PR modules are given in the column *Used slices*. The *Utilization* represents the percentage of slices in the PR modules that are utilized by the implementation. Configurations that could not be realized on the Spartan-3 FPGA because of the resource limitations of the PR-modules are marked with an $X$.

In Table 5.4 the execution time and the amount of frames per second are calculated starting from the write command for the first pixel of the frame to the final

**Table 5.4** Configuration test parameters for color recognition on BeBot

| N colors | N blocks | Used slices | Utilization | Execution time [$\mu$s] | Frame/sec |
|---|---|---|---|---|---|
| 1 | 1 | 1250 | 11% | 783.83 | 1275 |
| 1 | 2 | 1363 | 12% | 784.13 | 1275 |
| 1 | 4 | 1898 | 16% | 785.20 | 1273 |
| 1 | 8 | 2547 | 22% | 810.60 | 1233 |
| 1 | 16 | 3867 | 33% | 5281.64 | 189 |
| 1 | 32 | 6794 | 58% | 20450.22 | 48 |
| 2 | 1 | 2351 | 20% | 783.83 | 1275 |
| 2 | 2 | 2912 | 25% | 784.13 | 1275 |
| 2 | 4 | 3672 | 31% | 785.20 | 1273 |
| 2 | 8 | 5061 | 43% | 810.60 | 1233 |
| 2 | 16 | 7816 | 67% | 5281.64 | 189 |
| 2 | 32 | X | X | X | X |
| 4 | 1 | 5174 | 44% | 783.83 | 1275 |
| 4 | 2 | 6165 | 53% | 784.13 | 1275 |
| 4 | 4 | 7434 | 64% | 785.20 | 1273 |
| 4 | 8 | 9846 | 84% | 810.60 | 1233 |
| 4 | 16 | X | X | X | X |
| 4 | 32 | X | X | X | X |

interrupt provided by the module. At that time the results of the computation are already stored in the output FIFOs.

Reconfiguring between optical flow and color recognition requires 20 ms. The camera sends data with 30 frames per second, which results in 33 ms for calculation. Since the optical flow is calculated in 0.9 ms, 12.1 ms are available for color recognition if dynamic reconfiguration on a frame-by-frame basis is performed (cf. Fig. 5.64). Hence, optical flow and color recognition for up to 16 blocks can be performed virtually in parallel without frame-loss by dynamically reconfiguring between two frames.

The proposed hardware implementations of the vision algorithms on the BeBot miniature robot platform show that real-time image processing is possible even on platforms with limited processing capabilities. The use of FPGA-based hardware releases the processor from these very computational intensive tasks. Additionally, dynamic reconfiguration can be used to switch between different applications or to modify the elaboration parameters at run-time.

#### 5.4.5.3 Dynamic Reconfiguration of Multi Processor System on Chip

In addition to fine-grained FPGA-based architectures, coarse-grained architectures are evaluated. In this context, we focus on on-chip multiprocessors that integrate mechanisms for dynamic reconfiguration with minimum area overhead. In general, our MPSoC system comprises of a generic and hierarchical architecture, so

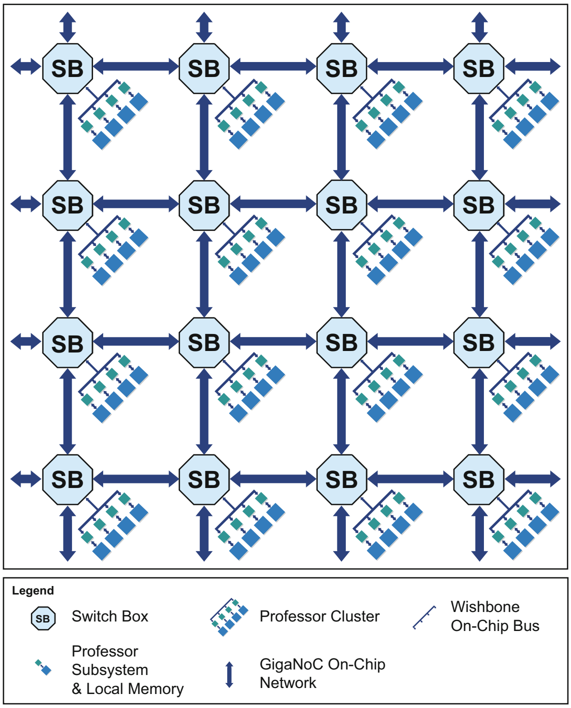


**Fig. 5.65** MPSoC architecture comprising the GigaNoC communication infrastructure

that the system can be configured for different application scenarios at design-time. Figure 5.65 depicts the MPSoC architecture proposed in [150], consisting of the *SoC level*, the *cluster level*, and the *processor level*. At the SoC level, a variable number of cluster components is connected via a network-on-chip communication infrastructure. Due to the homogeneous structure, the MPSoC system can be scaled to meet the performance requirements of various application domains. While the NoC provides the communication backbone for propagating data, at cluster level, this data is processed by a reconfigurable multiprocessor system. Via an on-chip Wishbone bus, the processor elements of each cluster can communicate locally and can access shared memory. A single processor element represents the lowest level of hierarchy of our MPSoC architecture.

GigaNoC is our hierarchical and scalable NoC communication infrastructure, which is especially suitable for multiprocessor SoCs [109, 170]. The GigaNoC architecture is depicted in Fig. 5.65. The switch boxes (*SB*) represent the core components of the NoC and act like high-performance routing nodes that propagate the data through the on-chip network. GigaNoC comprises of packet-switching [107] and each packet is divided into smaller fragments, called flits. In order to support arbitrary network topologies with different connectivity, the number of communication ports for each switch box can be configured during design-time. For the mesh topology, depicted in Fig. 5.65, every switch box has four external and one internal communication port, which connects the processor cluster to the NoC.

While the number of communication channels per switch box is chosen at design-time, the NoC topology and the routing strategies inside the switch boxes can be adapted at run-time. Several possible routing schemes are integrated into the hardware description of the switch box IP-core. A pre-selection can be made at design-time; at run-time the user or the operating system can easily switch between the integrated routing schemes by using special command flits. These command flits are also used to disable single malfunctioning embedded processors or complete processor clusters. In this case, the routing is automatically adapted to changes in the architecture.

QuadroCore Multiprocessor Cluster

Due to the generic implementation of the internal communication port, arbitrary processor cores and processor clusters can be attached to the NoC. Figure 5.65 depicts an example configuration, where clusters of four N-Core processors are attached to each switch box. N-Core is a 32-bit RISC microprocessor, which was developed in our group as a softmacro that can be easily adapted to the needs of specific areas of application [150]. N-Core has a common load/store architecture with a three-stage pipeline, which delivers reasonable performance for embedded systems.

The cluster organization based on the N-Core processor elements represents a typical MIMD multiprocessor cluster. In order to optimize flexibility and fault tolerance, a fast reconfiguration mechanism with low overhead has been added to the processor cluster, resulting in the run-time reconfigurable multiprocessor cluster *QuadroCore*. Without altering the instruction set architecture of the processors, run-time reconfigurability has been introduced by adding intra-processor interconnects to adapt the architecture in terms of synchronization, communication, and the degree of parallelism [105]. Figure 5.66 depicts the base architecture of QuadroCore and two typical configurations. Each of the four processors has its own local register file and instruction and data memory. Exchange of register contents between the four processors is achieved via a shared register file. Large amount of data sharing is possible via external shared memory, accessible by a shared bus.

The decision of altering the existing structure is driven by a special reconfiguration instruction, that has been added to the instruction set of the N-Core processors. This mechanism enables a quick, single-cycle run-time reconfiguration, i.e. very low overhead in terms of time required to reconfigure the resource connectivity. The reconfiguration instructions can be embedded into the normal program code by the programmer or by an optimizing compiler; no additional memory is required to store the configuration data, like in FPGAs.

In the proposed implementation, reconfiguration requires an alteration in the interconnection of the various building blocks inside and between the processors. Currently, capabilities for reconfiguration have been added between the decode & execute stages, between the execute & register read/write stages, and between the processors and the shared memory. In QuadroCore, the reconfigurable interconnect is realized by means of additional multiplexers that have been integrated into the architecture. The reconfiguration instruction provides the configuration information to determine the functionality of the reconfigurable interconnects between the intermediate stages of the instruction pipeline, i.e. the control signals of the added multiplexers.

As mentioned, the QuadroCore cluster can be dynamically reconfigured with respect to three main features: synchronization, communication, and parallelism, which are briefly described in the following paragraphs.

**Synchronization:** Depending on the amount and frequency of inter-processor data exchange, the processors in the cluster can operate synchronously at instruction level or asynchronously. The cluster can be adapted according to the application characteristics during run-time, since both a fine-grained synchronization scheme (for instruction-level parallelism) and a coarse-grained independent operation (for task-level parallelism) are supported. The run-time change in synchronization is achieved by introducing a synchronization instruction between parts of the application where a change in application characteristics is determined during compilation. The synchronous mode ensures a lock-step operation while the asynchronous mode initiates a barrier synchronization for every inter-processor data exchange.

**Communication**: The communication between the processing elements is mainly categorized in terms of frequency of data exchange and amount of data exchange. To suit applications where exchange of data such as register contents is frequent, a shared register file provides a quick data-exchange mechanism. For large amount of data, the shared memory is accessible via arbitration over a common bus. The shared register file has a round-trip time (write and read) of 4 clock cycles, whereas the shared memory has a variable access time between 5 to 12 clock cycles for each access. Furthermore, register sharing among processors is possible on account of the reconfigurable interconnect introduced between the ALUs and the register files as depicted in the right configuration in Fig. 5.66. For applications with high register pressure, registers from the neighboring processors can be utilized (supported by the compiler).

**Parallelism:** The choice of data-parallel or task-parallel behaviour steers architectural characteristics. The Multiple Instruction, Multiple Data (MIMD) mode, allows asynchronous operations on independent data and instruction streams. The Single Instruction, Multiple Data (SIMD) mode, as illustrated in the middle configuration in Fig. 5.66, co-ordinates all the four data-paths with a single instruction stream, thus saving energy via reduced memory interactions.

As mentioned above, the reconfiguration can be easily controlled by the user by adding simple reconfiguration instructions into the C code. A smarter way of introducing reconfiguration is to integrate the decision process into the compiler. As detailed in [105, 168], the choice of the best mode of execution can be made using standard program analysis techniques. For every basic block, our compiler, called COBRA (Compiler-Driven Dynamic reconfiguration of Architectural Variants), determines the best possible mode during compilation and a reconfiguration is inserted between the modes. The reconfiguration overhead is kept as low as possible since the reconfiguration between modes requires only a single clock cycle [167, 169].

### CoreVA VLIW Processor

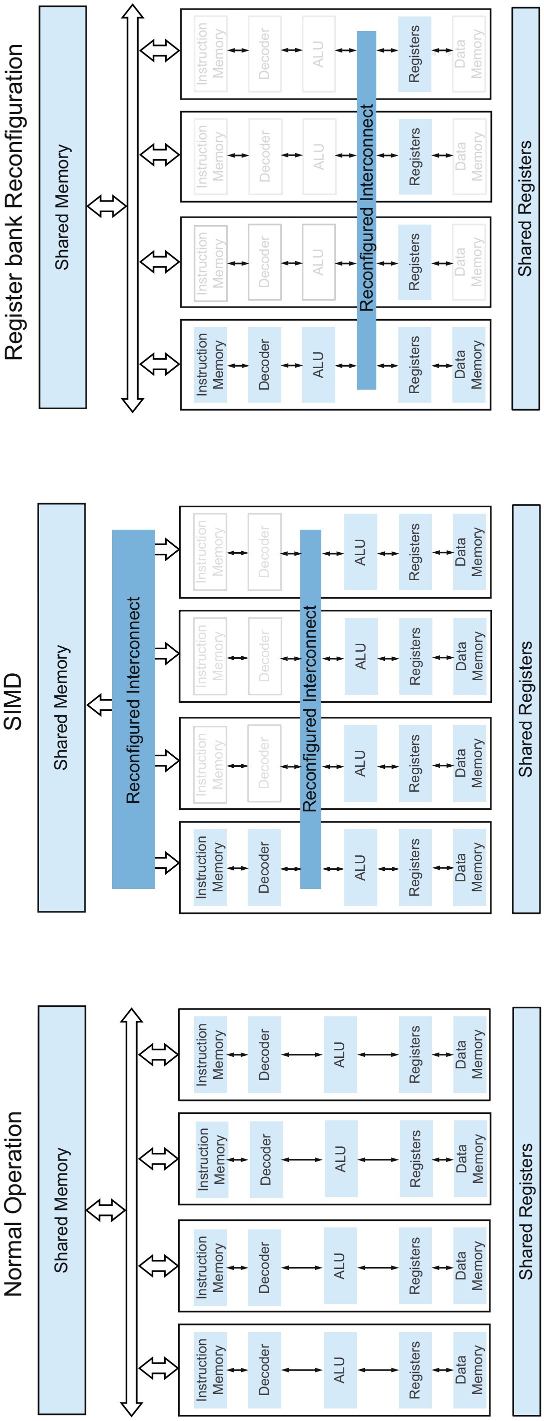
As an alternative to the QuadroCore architecture, we have developed a VLIW (Very Long Instruction Word) processor, especially suited for signal processing applications, called CoreVA [110]. Resource efficiency together with high flexibility were the main design goals for the processor implementation. The CoreVA architecture is a modular soft-core design, which can be configured at design-time with respect to the number of functional units (e.g. ALUs, Multiply-Acummulate (MAC) units, division step units), the width of the data paths, and the structure of forwarding circuits. In the default configuration, the CoreVA architecture represents a 4-issue VLIW architecture, implemented as a Harvard Architecture with separated instruction and data memory and six pipeline stages.

The operations follow a two- and three-address format and are all executed in one clock cycle. Most instructions have a latency of one clock cycle, except branch, MAC and load operations, which have a latency of two clock cycles. In SIMD mode, two 16-bit words can be processed in each functional unit, which leads to an eight-fold parallelism. As a first prototype, the CoreVA architecture has been fabricated in a 65nm STMicroelectronics technology. The CoreVA system (including level-1 cache and several dedicated hardware extensions) operates at a clock frequency of up to 285 MHz with a power consumption of about 100mW. The chip area is about 2.7sqmm including 32 kByte level-1 cache for instruction and data.

With respect to adaptability of the architecture, two mechanisms have been integrated to enhance the efficiency of the architecture: dynamic bypass reconfiguration and dynamic voltage and frequency scaling using a specially designed subthreshold standard cell library. The bypass reconfiguration exploits the fact that many paths of the integrated bypass are rarely used for certain applications. Therefore, the user can change between application-specific bypass configurations at run-time. Depending on the actual implementation, this leads to a reduction of the critical path by 26%, which can be used to dynamically increase the clock frequency or to decrease power consumption.

The next generation of CoreVA processor has been realized utilizing a specially developed ultra-low power standard cell library. The new processor, CoreVA ULP, is capable of dynamically adapting its operating parameters according to application requirements and environmental conditions at run-time [138]. During times of low processor load, power dissipation is substantially reduced by operating the processor in subthreshold mode. A chip containing two CoreVA ULP processors was fabricated in an STMicroelectronics 65 nm CMOS technology and has been successfully tested. At 1.2V, the average energy dissipation of a single-slot processor core is 110.22pJ (at a clock frequency of 100 MHz). The minimum energy point of 9.94pJ occurs at 325mV, i.e. energy savings of 11.1 % can be achieved during subthreshold operation. The average clock frequency at this point is 133kHz.

**Fig. 5.66** QuadroCore enables switching between different modes of operation at run-time



## 5.5 System Software

Stefan Groesbrink, Simon Oberthür, and Katharina Stahl

System software encompasses all software approaches that interconnect the application software layer and the hardware layer. This includes operating systems as well as other middleware approaches, e.g. virtualization.

In the context of self-optimizing mechatronic systems, the system software provides an execution platform for self-optimizing applications that run on online-reconfigurable hardware. The ability to cope with the dynamically changing requirements from either the software or the hardware and the capability to adapt to these changes at run-time is a prerequisite for system software being applied on self-optimizing mechatronic systems. Hence, the system software must implement self-optimizing methods by itself. Self-optimization in system software is thereby not restricted to reacting to dynamical changes. The system software may also implement methods that can be applied to self-optimize the performance of the execution platform concerning e.g. resource utilization.

The functionality the system software offers is strongly coupled to the requirements related to it. Referring to the overall design and development of the self-optimizing mechatronic system, the general requirements on the self-optimizing system software are defined by the principle solution (cf. Sect. 3.3.3). One example for a general requirement is the optimization of resource allocation by exploiting unused (however reserved) resources. This requirement is solved by means of the flexible resource management which is presented in Sect. 5.5.2. Another example for a general requirement on the system software is to support run-time dependability. The problem of run-time dependability is addressed by two different operating system service approaches, one is testing the application state by using an online model checking while the other one is inspired by artificial immune systems that tries to identify system behavior anomalies. However, dependability of self-optimizing mechatronic systems is it's own subject. These operating system services have been introduced in detail in [69, D.o.S.O.M.S. Sect. 3.2] .

General requirements express general properties of the system software, that is implemented in the form of an OS kernel module, an OS service, or a separate middleware layer.

However, being the interconnecting execution platform, there is a strong interdependency between the system software and the application software layer as well as between the system software and the reconfigurable hardware. Specific requirements arise from this interconnection in terms of provided interfaces or services that are required by the applications or on the other hand in providing abstractions of hardware implementation in order to encapsulate a change in the hardware configuration. Both, the general requirements on the system coming from the principle solution and the specific requirements arising from the software and hardware have to be satisfied by the system software so that an adequate platform for self-optimizing mechatronic systems can be ensured. We assume the self-optimizing system software to be composed of reusable components (cf. Sect. 3.3.3) that are

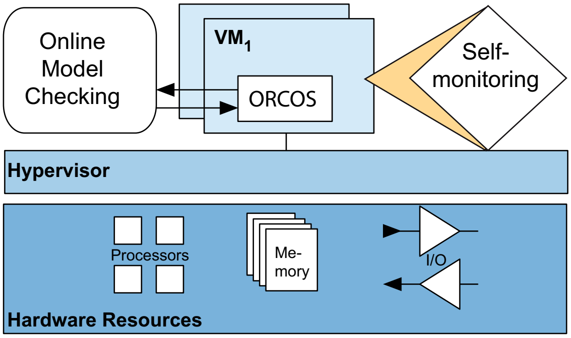


**Fig. 5.67** Self-optimizing system software layers

activated with respect to the present system requirements. Each system component addresses a specific system function or system property.

We separate the self-optimizing system software into two different layers: one containing the self-optimizing real-time operating system ORCOS and the lower level layer containing the virtualization platform including the hypervisor named Proteus. This section presents the self-optimizing real-time operating system ORCOS and the self-optimizing virtualization platform. Fig. 5.67 shows an overview of the system software architecture. In addition, figure Fig. 5.67 also illustrates how the operating system ORCOS integrates the methods for *Online Model Checking* and for *Self-Monitoring* which addresses the system's dependability.

Considering the **real-time operating system** (RTOS) of a self-optimizing mechatronic system, it has to be able to cope with dynamically changing system behavior and hence dynamical changes on the requirements of the platform. We distinguish between external and internal requirement changes. External requirement changes are those originating from outside the system software. That means changes in the requirements either from the software or the hardware layer. For example, due to a software reconfiguration, an OS service will be required that has not been provided by the operating system before. Another scenario might be that the implementation strategy of an OS service must be exchanged based on the new software configuration. The biggest challenge to cope within such a system is that not all system parameters are determined during design-time and therefore have to be identified during run-time. Resulting from this, the OS has to provide an interface that allows to signify the changes in requirements. Furthermore, to be able to satisfy these changes, the operating system must also provide alternatives in implementation. And last but not least, the operating system needs structures to enable the run-time activation or online exchange of components.

Internal requirements come from the operating system itself. As being a self-optimizing operating system, it is equipped with desired objectives and optimization criteria. Usually, the operating system's objective is to optimally manage the application's task and the resources, e.g. in terms of resource consumption, memory management, scheduling strategies, etc. To achieve those requirements, the operating system requires internal structures to monitor and analyze the performance and to verify whether the optimization objectives are fulfilled. Obviously, all self-optimization efforts of the operating system must be performed under considerations of real-time constraints.

The **real-time operating system ORCOS** [59] was first designed to be fully customizable during design-time. However, for the purpose of online self-optimization, we needed an operating system that is flexible and can be extended during run-time. Therefore, we adjusted the architecture of ORCOS to enable information processing required to enable self-optimization. We will describe the resulting OS architecture in Sect. 5.5.1. As a basis for run-time reconfiguration, we use the concept of our Profile Framework, described in Sect. 5.5.1.1, that allows alternative implementations for applications task as well as OS components.

Modern self-optimizing mechatronic systems have highly dynamic resource consumption. One of the main objectives of a real-time operating system is to optimize resource management. Common real-time systems and middleware software are fixed and not optimal for such scenarios. A problem with dynamic real-time applications using common real-time system software is that applications allocate resources up to their maximum requirements. On the one hand, this allocation behavior guarantees that the applications have all resources being required during execution. On the other hand, the maximum resources are often required only in the worst case and are mostly unused. An approach for optimizing the dynamic resource consumption is presented by applying flexible resource management. We developed a *Flexible Resource Manager* (FRM) (cf. Sect. 5.5.2) that allows to optimize resource consumption autonomously. It uses the Profile Framework as the basis for alternative resource requirements of applications. The FRM optimizes resource consumption in such a way that it allows temporal usage of resources that are reserved by other applications for worst-case. However, this overallocation of resources is conducted in a safe manner as the FRM guarantees a reconfiguration of the system without violating worst-case deadlines.

An additional implementation technique for self-optimization – especially in terms of resource utilization – is our **virtualization platform**. Like any kind of virtualization our approach provides strict separation of hardware resources by means of a hypervisor. A two-level scheduling (hypervisor and RTOS) has been designed in such a way that real-time aspects are strictly taken into consideration. The FRM approach has been extended in such a way that now a two-level FRM (integrated in hypervisor and RTOS) is implemented and dynamic reconfiguration may even happen across virtual machines (cf. Sect. 5.5.4). Some further interesting aspects concerning enhancing the system dependability by virtualization are illustrated in more details in [69, D.o.S.O.M.S. Sect. 3.2] .

## 5.5.1 Architecture for Self-optimizing Operating Systems

As the first step, we developed a real-time operating system named **ORCOS (Organic Re-Configurable Operating System**, see [59]) that allows a fine-grained configuration of the basic (functional) OS components according to the given requirements. According to the definition of self-optimization (cf. Sect. 1.2), the

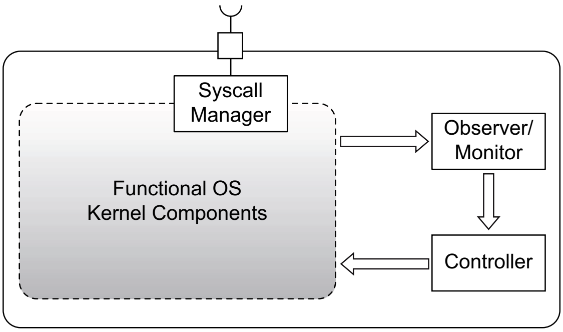


**Fig. 5.68** Architecture for a self-optimizing operating system

workflow of a self-optimizing system includes the following steps: Analysis of the current situation, determination of objectives and adaptation of the system behavior. Hence, the self-optimizing real-time operating system must provide mechanisms and structures to enable the implementation of a self-optimization workflow.

The architecture of the RTOS is required to be extended in order to support self-optimization and adaptation to the changing requirements of the self-optimizing software and hardware. We adjusted the OS architecture based upon the Observer-Controller Architecture that was first instantiated by the Organic Computing Initiative [179]. An Observer and a Controller component extend the OS and build up the basis of self-optimization and enable monitoring, self-reflection and reconfiguration in the real-time operating system. The resulting architecture of the operating system ORCOS is presented in Fig. 5.68. These new components are integrated into ORCOS as configurable kernel components but separated from the functional OS kernel modules. The Observer is responsible for monitoring and data collection, and analyzing and evaluating system behavior based on the defined system policies and objectives. Self-optimization in the operating system is triggered by a reconfiguration that is initiated by the Controller as a reaction on the evaluation procedure results. The ORCOS Profile Framework builds up the basis for reconfiguration in the operating system, as it offers alternative implementations defined within a profile from which the Controller can select.

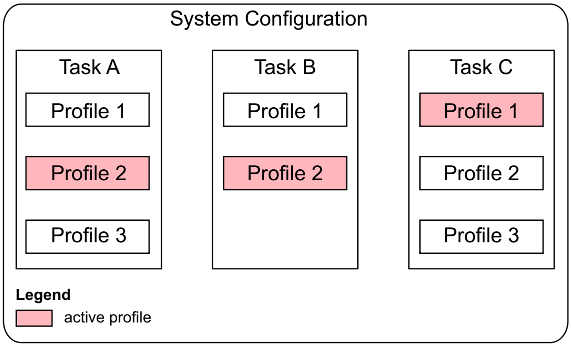
#### 5.5.1.1 Reconfiguration Framework

The central component for the run-time reconfiguration of the operating system is build up by the **Controller**. It is responsible for initializing a reconfiguration on the basis of the Profile Framework [156] provided by ORCOS.

Originally, it has been developed in the context of the Flexible Resource Manager (FRM) [154] to self-optimize resource consumption in resource restricted real-time systems as it allows for alternative implementations of an application task in terms of resource requirements.

The Profile Framework follows the following principle: at each point of time exactly one profile of a task is active (cf. Fig. 5.69). A configuration $c$ of the system

**Fig. 5.69** Profile Framework: Example for a system configuration



is defined as configuration $c = (p_1, p_2, \ldots, p_n)$, with $n$ being the number of running tasks $\tau$ and $p_1 \in P_1, p_2 \in P_2, \ldots, p_n \in P_n$ and $P_i$ being the profile set of task $\tau_i$. Each task must define at least one profile to be executed. For any task there may be available multiple profiles, i.e. versions with different parameters concerning nonfunctional properties. Selecting a specific profile is completed due to dynamic decisions at run-time.

We enlarge the concept of a profile to be applied to any reconfigurable OS component. This encompasses all system entities:

- application tasks
- OS kernel components and services
- components of the self-optimizing framework (e.g. Observer consisting of a reconfigurable Monitor and Analyzer described before)

Profiles may differ concerning their resource demands, which resources are applied (e.g. a specific communication resource), the implemented algorithm (e.g. in terms of accuracy of the algorithm or the strategy), execution times, deadlines etc. A prerequisite for identifying a component to be a reconfigurable component is the existence of alternatives, which in fact means the definition of at least two different profiles. Applying profiles to all OS components and the applications running on the system, all system parts become (re)-configurable online. In this case we can really speak about fully realizing a self-optimization operating system.

For reconfiguration the Controller contains policies, restrictions and thresholds for decision making. Of course, a decision for reconfiguration must be checked against the system characteristics and the real-time requirements of the application tasks. A reconfiguration must not harm the system service delivery and guarantee the compliance with the task's real-time deadline. If all the conditions are met, the Controller performs a reconfiguration of the system at run-time by simply switching between the profiles.

### Observer

The Observer component is responsible for (1) identifying and selecting the appropriate information and (2) analyzing them in order to make conclusions on future

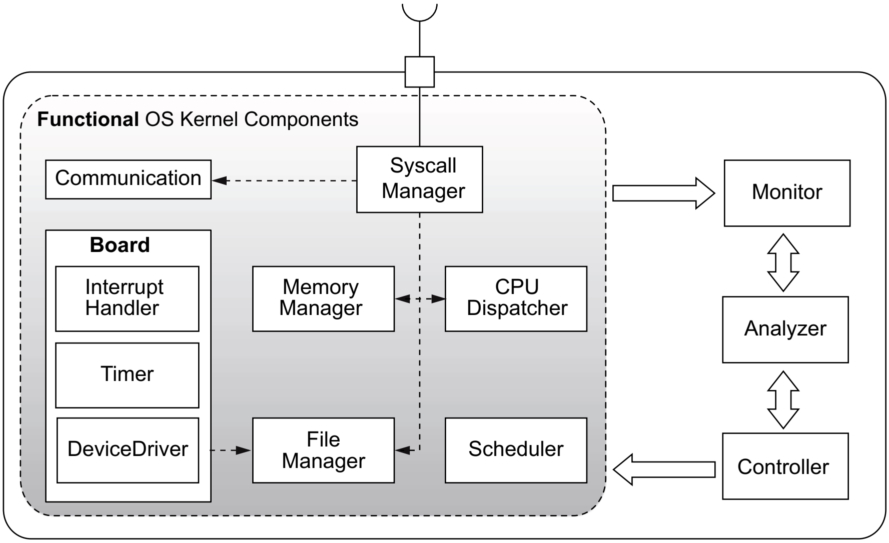


**Fig. 5.70** ORCOS Architecture integrating Monitor, Analyzer and Controller.

system states. For our purpose, we subdivide the observer into two separate entities: a Monitor and an Analyzer (cf. Fig. 5.70). Although, these two entities are strongly coupled (the monitor only collects the data that is required by the analyzing method) they exhibit self-contained tasks which can be executed in a timely decoupled manner. Due to the ability of online reconfiguration we introduce an additional interconnection between the components: The strategy of the analyzer is reconfigurable at run-time and can be exchanged by the controller. The data collected by the monitor depends on the applied analyzing algorithm so that an exchange of the analyzing strategy in turn has effect on the data aggregation of the monitor.

The monitor collects the behavioral data and aggregates the data for the analyzer while the analyzer evaluates the data and passes its evaluation results to the Controller. In order to collect data about the application behavior, we can monitor the SyscallManager interface as it is the only interface through which a task can interact with the kernel. Integrated into the SyscallManager, the Monitor is able to intercept and record all the system calls from the tasks.

The idea is to enable the use of a broad range of analysis algorithms with the same monitor. Hence, the monitor is designed to be *independent* of any analysis algorithm that evaluates behavior profiles. However, different analysis algorithms use different parameters to evaluate task behavior. Some of them use system calls and system call arguments, while others use the return address stack or the program counter [58].

The monitor defines different *monitor modes* to make it reconfigurable at run-time in order to modify which parameters are monitored in accordance to the analyzing algorithm and control monitor memory usage. The monitor is designed to selectively monitor specific parameters like the system call id, system call arguments, return address stack, task resources etc. These parameters can be reconfigured at run-time by changing the monitor's mode. By using this filtering mechanism, the monitor minimizes the overhead associated with monitoring data since it only aggregates the data that is required to build up the behavior knowledge base for the associated anomaly detection algorithm. To set up the monitor modes, the monitor offers an interface through a *monitor API*.

The monitor extracts the system call information parameters along with further OS state information whenever a system call happens. As the monitor intercepts the system call execution in order to record the data, it introduces additional run-time overhead into the system call handling.

Furthermore, real-time systems have limited memory and, hence, real-time applications have severe restrictions on the amount of memory they can use. The monitor has similar restrictions. System call information is collected at thread basis. The amount of memory used is also governed by the frequency of system calls invocations and the amount of monitored parameters. This mechanism enables the controller to re-configure the monitor at run-time in order to prevent memory overflows and safeguard overall memory usage.

Analyzer

The ORCOS Analyzer provides a framework for anomaly detection algorithms that, based on the run-time reconfigurability of the system, may be exchanged at run time. Algorithms implemented to analyze system behavior must comply with the API provided by the monitor and be applicable for analyzing self-x behavior.

The workflow of each analyzing algorithm is:

1. generate a behavior representation according to the requirements of the algorithm from the behavioral data base provided by the monitor,
2. evaluate and match the actual behavior against the normal behavior profile provided knowledge base,
3. inform the controller in case of deviations and detected anomalies.

Strictly separated from the monitor which in turn is directly attached to the system call handler, the analyzer is scheduled individually. The monitor must record the data whenever a system call is invoked while the analyzer is triggered when a sufficient amount of behavioral data is available (determined by the anomaly detection strategy). An anomaly detection method that can be integrated into the analyzer is introduced in [69, D.o.S.O.M.S. Sect. 3.2] .

## *5.5.2 Self-optimized Flexible Resource Management*

The Flexible Resource Manager (FRM) [155] permits an over-allocation of resources under hard real-time constraints. The technique allows to minimize the internal waste of resources by putting temporarily unused resources, which are only reserved for the worst-case, at other applications' disposal. Additionally, an adaptive self-optimizing system or middleware software can be built using this technique.

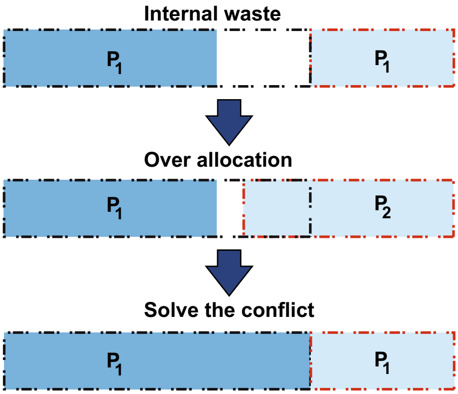
To use the Flexible Resource Manager, applications have to stick to a specific resource allocation paradigm and can specify multiple modes of operation – so called profiles – to allocate additional resources if other applications temporarily do not need them. The resource allocation paradigm comprises:

1. The application has to specify *a priori* the minimum and maximum limits per *resource usage*. The application can not acquire less or more resources than specified in the current active profile, which the FRM activates. If the application wants to do so, then it has to specify a new profile with appropriate limits. The activation of the new profile underlies an *acceptance test* of the FRM.
2. The FRM is in charge of the assignment of applications into their profiles. If a reconfiguration between profiles is enforced by the FRM, application-specific transition functions are activated. This allows for an application-specific change between different operation modes with different resource requirements.
3. The FRM also registers the actual resource consumption of the active profile of an application, which must be within the specified limits. The FRM guarantees to the applications that they can allocate the resources up to the specified limit in the active profiles. In case of a resource conflict – when the system is over-allocated – the FRM solves the conflict by forcing applications into other profiles so that every resource request can be fulfilled. The FRM ensures that no deadlines of hard real-time tasks are violated. This is done by only allowing an over-allocation of a resource if a plan for solving every possible conflict exists and this plan can be scheduled under hard real-time constrains. Figure 5.71 illustrates this approach.
4. Resources are distinguished which can be reassigned within an negligible reallocation time and resources which have to be configured in the background by the system software. Resources which are reconfigurable in background need more time to be reassigned between different applications. All resource demands of background reconfigurable resources – also within the specified limits of the actual profile – require an announcement to the operating system. Between the announcement and the assignment a delay is assumed. The profile specifies a *maximum* delay per background reconfigurable resource. Note that this delay is a worst-case value.

The ability to schedule and the deadlock-freeness of the FRM approach have been formally proven.

To enable engineers to easily use the FRM and the profile model, the approach is integrated into the high-level design process for self-optimizing mechatronic systems. A semi-automatic code generation was presented [33], which allows for a generation of profiles out of hybrid real-time state charts. Hybrid state charts combine continuous models and discrete real-time state charts (e.g. for the reconfiguration model). The application programmer only has to specify a minimum of additional information to generate profiles. For simulation purposes the FRM was not only implemented on top of the operating system DREAMS [51] , but also integrated into MATLAB/SIMULINK. This enabled a simulation of an application using the FRM in which the continuous part, including the controller and the plant, of the application is encapsulated [157].

Fig. 5.71 Over allocation



## 5.5.3 *Self-optimization in the Operating System*

If self-optimizing applications change their behavior and their resource requests dynamically during run-time, even the underlying real-time operating system (RTOS) should reconfigure its QoS by means of the currently provided services. For example, a specific protocol stack should only be present in the RTOS, when applications request this protocol for their communication. I.e., a reconfigurable/customizable RTOS includes only those services that are currently required by its applications. Hence, services of the RTOS must be loaded or removed on demand. Thus, the RTOS also releases valuable resources that can be used by applications.

As self-optimizing applications are – in the context of this book – embedded mechatronic systems, they run under hard or soft real-time constraints. Thus, the reconfiguration of RTOS components is critical. The RTOS always has to assure a timely and functional correct behavior and has to support the required services. Hence, the reconfiguration underlies the same deadlines as the normal operation of the applications. To handle exactly this problem, the FRM can be applied to the RTOS as well. The FRM model executes the reconfiguration under real-time constraints. The acceptance test inside the FRM assures that the reconfiguration does not violate real-time constraints.

The main idea is to release resources of system services by deactivating/activating basic versions or activating development alternatives (e.g. an implementation on the FPGA instead on the CPU) of these services. These different states of the services are modeled as different profiles for each service. Then, these RTOS components will be handled by the FRM as normal application profiles. Thus, no change of the FRM model is required.

### 5.5.3.1 Reconfiguration Model

To build an online configurable RTOS, components which can be re-configured (activated, deactivated, etc.) have to be identified during run-time. For this purpose the offline configurator TEReCS is reused (**T**ools for **E**mbedded **Re**al-Time

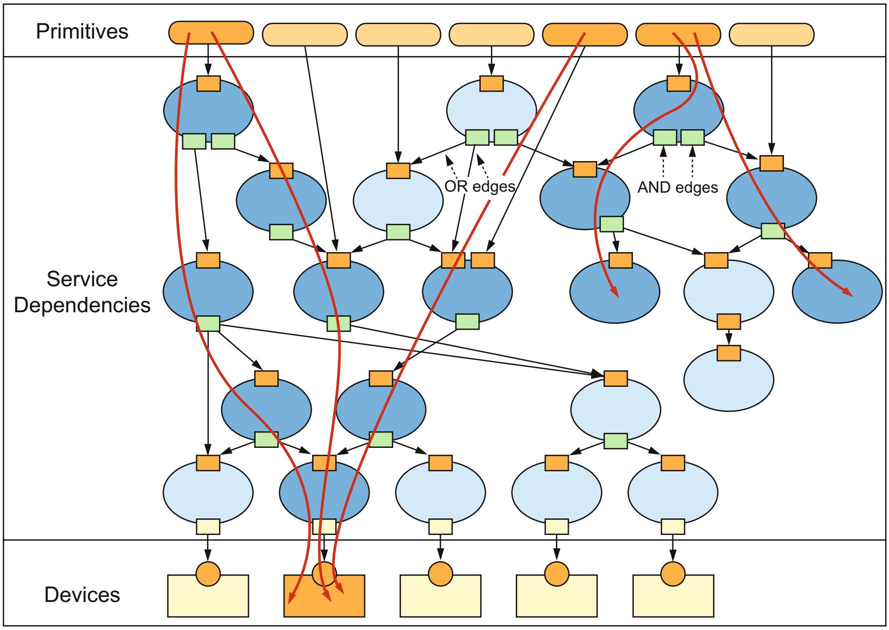


**Fig. 5.72** TEReCS's design space description from system primitives via services down to hardware devices (from [29]).

**C**ommunication **S**ystems) [28, 29]. In the TEReCS approach the complete and valid design space of the customizable operating system is specified in a knowledge base by a so-called AND/OR service dependency graph [36]. This domain knowledge contains options, costs, and constraints and defines an over-specification by containing alternative options.

The complete valid design space of the configurable operating system is specified by an AND/OR graph:

- Nodes represent *services* of the operating system and are the smallest atomic items, which are subject of the configuration,
- Mandatory dependencies between services are specified by the AND edges,
- Optional or alternative dependencies between services are specified by the OR edges,
- Services and their dependencies have costs and can be prioritized,
- *Constraints* (preferences, prohibitions, enforcements under specific conditions) for the alternatives can be specified,
- Root nodes of the graph are interpreted as *system primitives*/*system calls* of the operating system.

The algorithm works, e. g. for communication primitives, as follows: A path can be found through the complete graph from the sending primitive down to the sending device, considering the routing and then up to the receiving primitive. The services that are visited on this path have to be installed on the appropriate nodes of the service platform (see more color saturated nodes in Fig. 5.72). Thereby, the path should create minimal costs by the use of the services.

Such paths will be searched for all primitives that are used in the requirement specification. Because only a subset of all primitives is normally used, especially the particular selection is responsible for the instantiated services and its parameterization. The primitives can be considered as the strings of a puppet. Depending on which strings are pulled, the "configuration" of the puppet will change accordingly. The service dependencies can be compared to the joints of the puppet. Therefore, the algorithm is named *"Puppet Configuration"*.

The online configuration makes use of pre-defined solutions that have been configured offline. Thus, it is up to the online configuration phase to identify the use cases, for which the solutions have been created and to activate them. The identification is simple, because it depends on the system primitives, which are used by applications and other clusters. Those pre-defined solutions have to be instantiated so that all required primitives are implemented for the concrete situation during runtime. If primitives are unused an alternative cluster can be activated during run-time, which does not implement the unused primitives.

The same system primitives that have been used to create a pre-defined solution are leading to the selection of that solution component in the coarse-grained design space level. This condition must be assured during the specification of the abstract design space for the pre-defined solutions. This problem must also be solved by the system expert offline. This procedure is allowed, as TEReCS' main philosophy obliges the encapsulation of all expert knowledge in the design space descriptions.

Example

Figures 5.73 and 5.74 sketch an example for two pre-defined cluster options (B+C). The primitives *Scheduling* and *Communication* in Fig. 5.73 are used by the equally named clusters from Fig. 5.74. The option B is generated from A if the primitive *Scheduling* is not used. The option C is generated alternatively. In Fig. 5.74, the pre-generated solutions B+C are included as the *OS Hierarchy Option I* and *II*.

Except the cluster *Scheduling* all other clusters can use both options alternately. Only the cluster *Scheduling* requires explicitly the solution of the *OS Hierarchy II*, which supports multiple threads. If the primitive *CreateThread* will be used, then the cluster *Scheduling* is requested. Thus, the request of the primitive *CreateThread* from an application requests the cluster *Scheduling* to be instantiated. Moreover, as the cluster *Scheduling* requires the internal primitive *Scheduling*, the cluster *OS Hierarchy II* also has to be instantiated instead of the cluster *OS Hierarchy I*.

The alternatives of the operating system services are modeled as different profiles. Using the previous example, *OS Hierarchy I* and *OS Hierarchy II* are mapped into two profiles of the system service *OS Hierarchy*, which is from the point of view of the FRM handled as a normal application.

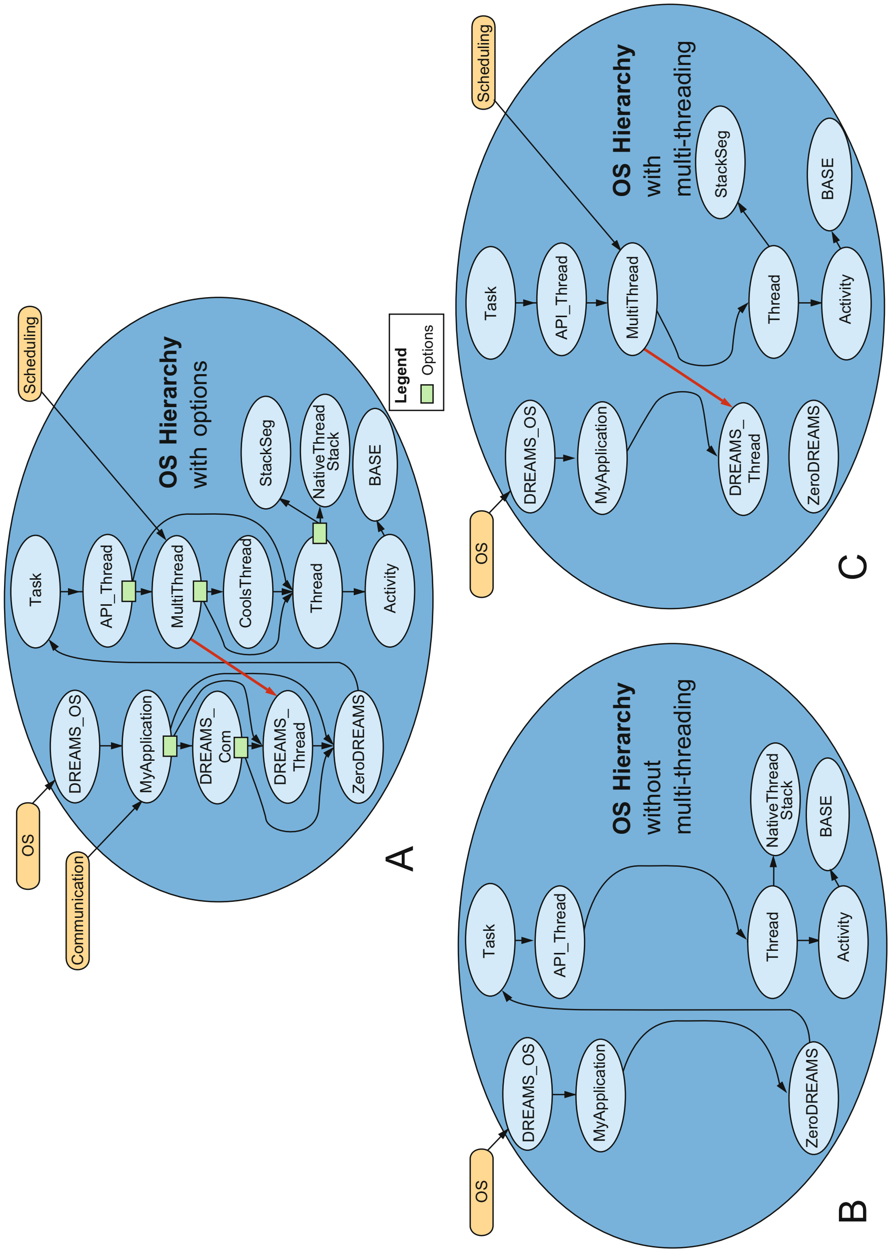


**Fig. 5.73** Zoom to the fine-grained level of to the OS cluster with its optional components (A) and two pre-defined configuration examples (B+C).

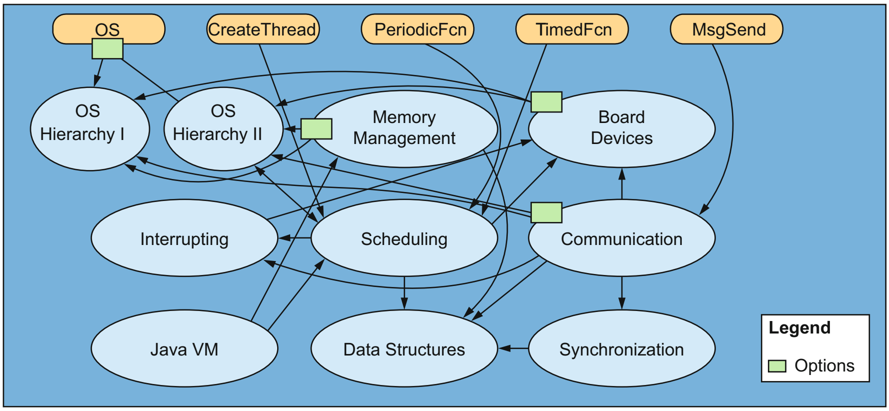


**Fig. 5.74** OS design space at $2^{nd}$ level with integrated options for pre-defined solutions of clusters.

### 5.5.3.2 Online Reconfiguration

For each primitive a new resource is introduced. When an application or other RTOS service wants to use a system primitive, it requests the corresponding resource. Initially, each service holds each corresponding resource of the primitive it provides. When an application or other system service arrives or wants to use a primitive, it has to request the corresponding resource, which must be in the range of the specified resource boundary of its actual profile. As a reaction, the FRM activates a corresponding profile of the service, where the service does not block the primitive by occupying the corresponding resource but implements the primitive by activating an alternative pre-defined solution. The service implements in the enter and leave functions of the profiles the switch between the pre-defined clusters. These reconfiguration functions represent the Online-TEReCS module as a whole entity. In a profile the meaning for the system services of holding a primitive provides the reverse meaning of an application: a service holding a primitive means that the primitive is not required and does not need to be implemented. On the other hand, when an application holds a primitive, the service has to provide the primitive's code.

Clusters representing the system services are reconfigured during run-time employing the FRM approach to mediate the reconfiguration. The alternatives are modeled as different profiles. Using the previous example again, *OS Hierarchy I* and *OS Hierarchy II* are mapped into two profiles of the system service *OS Hierarchy*, which is from the point of view of the FRM handled as a normal application.

As sketched in Fig. 5.75 and before, the reconfiguration of the RTOS cluster components is completely managed by the Online-TEReCS module in the enter/leave functions of the corresponding profiles. The reconfiguration options are modeled as optional profiles being offered by Online-TEReCS, which are activated and deactivated by the FRM. Each profile defines which primitives are used by a system service profile and which are not used – as the primitives are modeled as resources.

Thus, the FRM does not need to distinguish between RTOS components and normal applications. The FRM mediates the system primitives (resources) between all the applications and the RTOS. Thus, it handles the competition between the applications and the reconfiguration options of the RTOS. The system primitives represent the dependencies between the services. The services which are locked do not provide system primitives in corresponding profiles by allocating the system primitive itself. Thus, the FRM manages and assures that all dependencies are considered during run-time, otherwise the concrete allocation of a system primitive corresponding recourse would by surmount the maximum available number and lead to unfeasible system configuration. Real-time constraints are respected by modeling the reconfiguration time of the RTOS in the switching conditions respective to the minimum dwell time of the profiles and the acceptance test in the FRM.

Applications must define all real-time constraints regarding their future resource allocations. Additionally, an application can only allocate resources in the range of the profile, which is currently active. With this information the FRM guarantees by means of the acceptance test, that all resource allocations can be timely performed. Deactivating a system service, by activating a profile in which this service is not configured into the system, and an application, which is currently not using the service but specifying a possible future use through the defined profile parameters, creates an over-allocation state. If the application wants to use the service this leads into a reconfiguration. The acceptance test assures that a system service is only deactivated if the reconfiguration to reactivate the service can be executed "in time" to provide the resources when needed.

The creation of pre-defined solutions for clusters is done automatically. For each combination of possible requests or dismissals of *system primitives* and *internal primitives* a configuration is generated. For the optimization and reduction of the design space of the operating system, a system expert might restrict the combinations of parallelly instantiated system primitives to only those ones that make sense which cover other solutions and – with high probability – are not used simultaneously.

A repository stores all pre-defined solutions of the clusters. A cache will temporarily store the code and the description of optional configurations for clusters, in order to speed up the loading of required cluster implementations. The cache can retrieve other configuration's implementations from background storage (hard disk) or from the network (cf. Fig. 5.75).

The FRM tries to optimize the system according to the current resource requirements of the components (system services and applications) and the quality information of the profiles. To do this, the FRM requests the application and system services to change their current profiles. This results in a reconfiguration of the RTOS and an optimization of the resource usage between the applications and the operating system.

The FRM approach includes the definition of quality values per profile. Thus, the FRM can not only reason about the optimality of application profiles, but it can additionally reason about the optimality of the RTOS configuration.

By the integration of TEReCS and the FRM into a RTOS a self-optimizing real-time operating system (SO-RTOS) is derived. Such an OS adapts itself with the help of the FRM to the needs of the current applications executed on top of it. Using this technique services can be deactivated and freed resource can be put at the applications' disposal. The real-time capability of the FRM ensures that only such services are deactivated, which can be reactivated under hard-real time constraints, if required by application tasks.

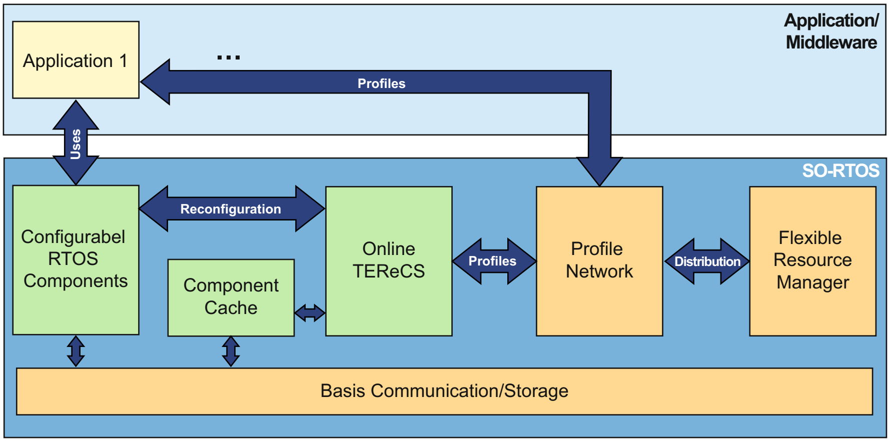


**Fig. 5.75** Integration of TEReCS and the FRM framework into the RTOS

### *5.5.4 Hierarchical Flexible Resource Manager*

The Flexible Resource Manager concept was adapted to system virtualization [195]. Integrated systems with multiple software systems executed on a single hardware unit provide often a more resource-efficient implementation compared to multiple separated hardware systems. System virtualization realizes this integration of multiple systems with maintained separation, and therefore is well suited for safety-critical mechatronic systems. The hypervisor allows the sharing of the underlying hardware among multiple operating systems, each executed in an isolated virtual machine. We developed a real-time capable hypervisor for embedded systems, which is characterized by multicore support and possibility to host both paravirtualized and fully virtualized guests [14, 77].

The Hierarchical Flexible Resource Manager consists of FRM components on two levels, *Guest-FRMs* and *Hypervisor-FRM*, as depicted in Fig. 5.76. In a partitioned manner, the FRMs on both levels take resource management decisions. The Guest-FRM is part of the operating system and switches between task profiles in order to assign resources to tasks, as previously described in detail. The Hypervisor-FRM is part of the hypervisor and switches between virtual machine profiles in order to assign resources to virtual machines. Communication takes place in both directions. The Guest-FRMs inform the Hypervisor-FRM about the dynamic resource

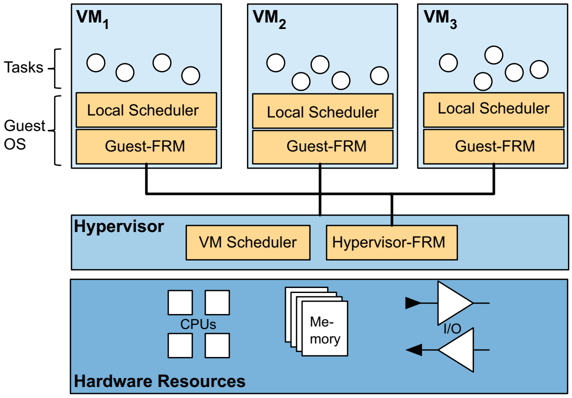


**Fig. 5.76** General Architecture: Hierarchical Flexible Resource Manager [195]

requirements and current resource utilization. The Hypervisor-FRM's resource allocation among the virtual machines is based on this information. The Hypervisor-FRM informs the Guest-FRMs about the assigned resources in order to allow the Guest-FRM to manage its resource share. The cooperation of the hypervisor's virtual machine scheduler and Guest-FRM guarantees that each guest system becomes (1) active in time, (2) for a sufficient duration and (3) equipped with the necessary resources, in order to allow the guest to execute its applications in compliance with their timing requirements.

The implementation of the Hierarchical Flexible Resource Manager requires paravirtualization, since the Guest-FRMs as part of the guest operating systems have to pass information to the Hypervisor-FRM as part of the hypervisor. According to paravirtualization [16], the guest operating systems are aware of being executed in a virtualized manner on top of a hypervisor and not on top of the bare hardware. The guest operating systems are modified and explicitly ported to the interface of the hypervisor. By consequence, they are able to communicate with the hypervisor. The requirement to modify the guest operating system is outweighed by the advantages gained in terms of flexibility of an explicit communication and cooperation of hypervisor and operating systems.

In contrast to static virtualization techniques where the resource shares are assigned a priori to the virtual machines, our approach allows for a dynamic resource allocation even across virtual machine borders. The cooperation of Hypervisor-FRM and Guest-FRMs is based on a hierarchical mode change protocol. We refer to profiles and transitions between them. A non-empty set of *task profiles* is assigned to each task, as introduced before. The Guest-FRM is in charge of switching between these profiles at run-time. The task profile $P_{r_j}$ of task $\tau_j$ is defined as:

- resource allocation minimums and maximums:
  $\forall\ resources\ R_k\ with\ limit\ \hat{R}_k : 0 \leq \phi_{j,k}^{min} \leq \phi_{j,k}^{max} \leq \hat{R}_k$
- profile quality $Q(\tau_j) \in [0,1]$
- subset of the set of task profiles to which the Guest-FRM can switch from $P_{r_j}$

In addition to task profiles, there are *VM profiles*, which specify the minimal and maximal resource limits for a virtual machine. VM profiles unite the active profiles of the tasks of a virtual machine. In case of a task profile transition, the VM profile is updated and communicated to the Hypervisor-FRM and used for the resource assignment among the virtual machines.

A VM profile $P_{VM_i}$ is defined as follows:

- resource allocation minimums and maximums:
  $\forall\ resources\ R_k : \forall\ tasks\ j\ of\ VM_i :\ \Phi^{min}_{i,k} = \sum_j \phi^{min}_{j,k}\,,\quad \Phi^{max}_{i,k} = \sum_j \phi^{max}_{j,k}$
- profile quality $Q(VM_i) \in \mathbb{N} : Q(VM_i) = \sum_j Q(\tau_j)$
- subset of the set of VM profiles to which the Hypervisor-FRM can switch from $P_{VM_i}$

The set of active profiles is called *configuration*. The possibility to switch between profiles on both the task level and on virtual machine level enables a dynamic resource assignment across virtual machine borders. A Guest-FRM can shift resources by task profile switches from one task to another; and similarly, due to the cooperation of the FRMs on the two levels, resources can be reallocated from task $\tau_i$ of $VM_1$ to task $\tau_k$ of $VM_2$. The Hypervisor-FRM activates a VM profile with a lower resource allocation maximum for $VM_1$ and according to this, the Guest-FRM of $VM_1$ activates a task profile with a lower resource allocation for $\tau_i$. This allows the Hypervisor-FRM to activate a VM profile with a higher resource allocation maximum for $VM_2$ and the Guest-level FRM of $VM_2$ to activate a task profile with a higher resource allocation for $\tau_k$.

The hierarchical FRM assigns fractions of resources at run-time to other tasks whenever a task does not use the complete amount of resources as needed in the worst case. If at a later point in time, the resource lending task needs more resources than remaining, a *resource conflict* occurs and has to be solved under real-time constraints. There are two kinds of resource conflicts, caused by two kinds of dynamic resource reallocation. The Guest-FRMs can reallocate resources among their tasks and the Hypervisor-FRM can reallocate resources among virtual machines. In both cases, an acceptance test precedes and a resource reallocation is accepted if and only if:

- $\forall\ Resources\ R_k\ ,\ \forall\ tasks\ 1..n : \sum_{i=1}^{n} \phi^{max}_{i,k} \leq \hat{R}_k$
- the FRM identifies a feasible *reconfiguration*

A reconfiguration is a sequence of profile switches that activate a configuration, which fulfills the worst-case requirements of all tasks. If such a reconfiguration plan includes VM profile switches, it is called *global reconfiguration*. The Hypervisor-FRM and at least two Guest-FRMs have to perform configuration switches. In contrast, a *local reconfiguration* only includes task profile switches and is accomplished by a single Guest-FRM. A reconfiguration plan can only be accepted, if the schedulability analysis attested that the time required to execute the reconfiguration does not lead to a deadline miss. The reconfiguration plans for conflict resolution are stored in *conflict resolution tables*. An entry is created after a reconfiguration was accepted and lists the required profile switches to reach a state that guarantees all

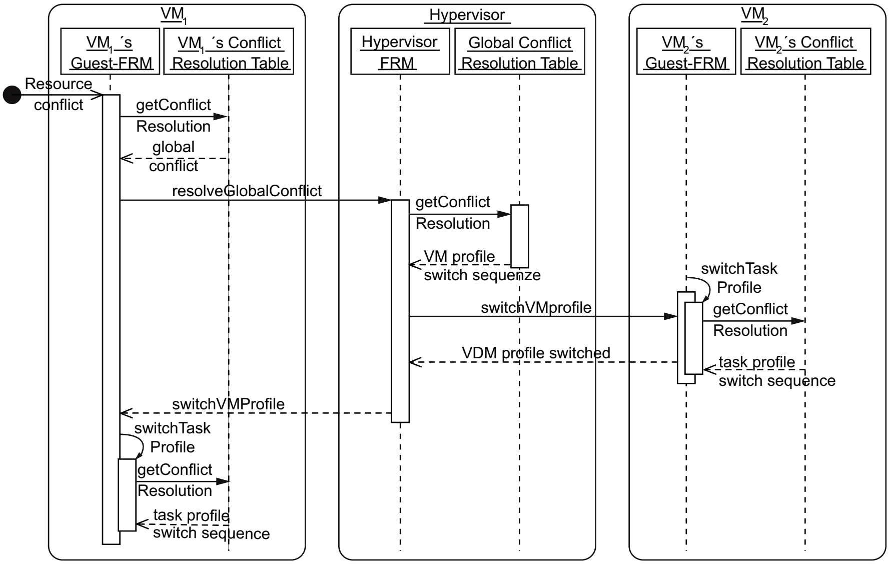


**Fig. 5.77** Conflict Resolution: Global Reconfiguration Sequence [195]

deadlines. If a conflict is always solved by reconfiguration to the initial state, there is at most one entry per task profile. A larger table with the possibility to reconfigure to multiple optimization levels is more promising, but requires additional memory.

In the following, an example depicts the conflict resolution process. It is assumed that task $A$, executed in virtual machine $VM_1$, has a specific worst-case requirement of a resource and consequently, this resource share was assigned. Since the actual resource usage of task $A$ was significantly below the reserved amount, the Guest-FRM switched to another profile and made a fraction of the assigned resources available to task $B$ of the same VM. In case of a resource conflict, i.e. task $A$ requires a larger resource share than remaining, the Guest-FRM resolves the conflict by switching to task profiles with a resource distribution that fulfills the timing requirements of task $A$. The sequence of profile switches that have to be performed to obtain this state was stored in $VM_1$'s local conflict resolution table when the acceptance test for the resource reallocation was passed.

It is possible that the Guest-FRM can not resolve the resource conflict, since a global reconfiguration is required to achieve this. This is the case, if it was caused by a resource reallocation to another guest system. A share of the resource reserved for virtual machine $VM_1$ could have been assigned to virtual machine $VM_2$ by the Hypervisor-FRM, and further assigned by $VM_2$'s Guest-FRM to task $C$. The Hypervisor-FRM informed $VM_1$'s Guest-FRM about this resource reallocation and the Guest-FRM noted this in the local conflict resolution table. The conflict resolution is depicted in the UML sequence diagram of Fig. 5.77. In case of a resource conflict of task $A$, the Guest-FRM of $VM_1$ informs the Hypervisor-FRM, which prompts the Guest-FRM of $VM_2$ to release the supplemental resources. The Guest-FRM of $VM_2$ switches the profile of task $C$ to one of lower resource utilization. The Hypervisor-FRM can accordingly switch the profile of both $VM_1$ and $VM_2$ and inform the Guest-FRM of $VM_1$ to ultimately activate the conflict resolving profile switch for task $A$.

In order to guarantee real-time requirements, hypervisors for embedded systems typically assign virtual machines statically to processors. This static approach is inappropriate for the varying resource requirements of self-optimizing mechatronic systems. Virtualization's architectural abstraction and the encapsulation of virtual machines support migration, i.e. the relocation of a virtual machine from one processing element and connected memory to another one at run-time. Prerequisite is a multiprocessor architecture with an instance of the hypervisor running on each processor. Multiple processing elements operate on their own dedicated memory, but are connected by input/output devices.

By the application of emulation, virtual machine migration is even possible for heterogeneous multiprocessor platforms, which are characterized by processors with differing instruction set architectures. Emulation executes program binaries that were compiled for a different architecture. This translation between instruction set architectures realizes cross-platform software portability. We developed a real-time capable emulation approach [115] and a real-time migration for heterogeneous multiprocessor architectures, which analyzes at run-time whether a virtual machine with real-time constraints can be performed without risking a deadline miss [82]. It selects an appropriate target for the migration and controls the migration process. This migration manager was integrated into our hypervisor Proteus [83].

For the migration of virtual machines, a coarse-grained dynamic reassignment of resources can be realized in comparison to the mode switches of the Hierarchical Flexible Resource Manager. In particular it is useful for open systems, in which the addition of applications or subsystems at run-time is possible. System virtualization isolates arriving potentially faulty or malicious software from existing critical applications. The acceptance of an application or even an entire subsystem typically changes the load balancing significantly and it might actually be necessary to perform migration to be able to accept an arriving subsystem.

System virtualization and its ability to reuse subsystems is a powerful technique to meet the functionality and reliability requirements (see [69, D.o.S.O.M.S. Sect. 3.2.8] ) of increasingly complex systems and has potential to support the migration to multiprocessor platforms. Targeting this architecture, the Hierarchical Flexible Resource Manager provides a resource management for the dynamically varying resource requirements of integrated adaptive systems. The two-level solution beyond virtual machine borders has the potential to increase the resource utilization significantly compared to static approaches.

## 5.6 Virtual Prototyping

Jörg Stöcklein, Wolfgang Müller, Tao Xie, and Rafael Radkowski

Virtual prototyping is a technique, which applies Virtual Reality-based product development for the engineering of mechanical and mechatronic systems. Virtual prototyping is based on the modeling, design, and analysis of Virtual Prototypes (VPs), i. e. computer-internal models, which are executed and analyzed in a Virtual Environment (VE). VPs are typically developed prior to physical prototypes (or mockups), which are mainly profitable for relatively small subsystems, e. g. the Hybrid Energy Storage System (HES) (cf. Sect. 2.1.5) or Active Guidance Module (cf. Sect. 2.1.3). Compared to physical prototypes, the development of VPs is less expensive and time-consuming, and VPs provide a significantly higher flexibility for change requests and variant management. Moreover, due to the virtualization of the prototype and the environment, Virtual Prototypes faciliate the early evaluation of the final product. All experiments can be conducted under controlled conditions of a well structured Virtual Test Bench (VTB) and for instance can easily be repeated for regression testing.

Today, with the outcome of sufficiently fast and affordable computing platforms and devices, **Virtual Reality** (VR) based Virtual Prototyping is widely accepted in several engineering disciplines. Examples are the design review of vehicles [114, 192] and plant engineering [212]. Meanwhile, **Augmented Reality** (AR) based technologies also are frequently considered for engineering related applications such as evaluation of automotive prototypes and the preparation of experiments [117, 209].

We apply Virtual Prototyping for the development of complex self-optimizing mechatronic systems with inherent intelligence, which react autonomously and are flexible to changing environmental conditions. This applies at each level of the hierarchical structure that makes up a complex mechatronic system: e.g. Mechatronic Function Modules (MFM) such as an intelligent suspension strut, Autonomous Mechatronic Systems (AMS) such as a vehicle, and Networked Mechatronic Systems (NMS) such as a vehicle convoy. Due to their complex structure and highly dynamically evolving behavior, self-optimizing mechanical systems impose huge challenges during their entire product development process, starting from the initial specification to composition, analysis, testing, and final operation [68].

As such, for the development of self-optimizing mechatronic systems, the adequate combination of VTBs, VEs, and simulation-based VPs can be beneficial over classical development platforms as they are highly flexible and customizable for the individual, dynamic needs and requirements of such systems and support different views and seamless integration of multiple integrated models. Nevertheless, the manual configuration of virtual platforms for mechatronic systems is cumbersome and error-prone as it is conducted on an individual base and requires the integration of different domains like electrical and mechanical engineering.

We introduce a novel Virtual Prototyping platform dedicated to the development of self-optimizing mechatronic systems. The platform seamlessly combines

VR- and AR-based user interaction and control with model-based execution of the different integrated VPs, which are controlled by domain-specific simulators or integrated Hardware-in-the-Loop components [17, 172]. For true design automation, we investigated automatic linking of the different models to VEs and advanced VTB technologies for the controlled and repeatable execution of experiments.

The remainder of this section is organized as follows. In the next subsection we will introduce the principles of VPs and VEs 5.6.1. Next we will describe our approach to automatic VE configuration. Section 5.6.2 presents the agent-based automatic model linking. Section 5.6.3 gives an agent-based solution to link the models to different visualizations. Finally, Sect. 5.6.4 outlines the basic concepts of our self-optimizing VTB, which is based on the principles of mutation analysis. All subsections present application examples based on the reconfigurable miniature robot BeBot [99], which serves as one of our main development platforms.

### *5.6.1 Virtual Prototypes and Virtual Environments*

Rafael Radkowski

A **Virtual Prototype** (VP) is defined as a computer-internal representation of a real prototype of a product [128]. Figure 5.78 outlines the basic concept of a VP, which is based on the notion of a digital mock-up (DMU) with the definition of the product shape and structure. A DMU is typically based on two models: 3D CAD models and the logical product structure. A VP extends a DMU by further domain-specific aspects like the kinematics, dynamics, force or information processing. Each of these aspects is defined by a different domain-specific view. As such, VPs help engineers to exercise, analyze, and evaluate the interaction of the system and its subcomponents. That way, VPs facilitate an easy comprehension of the product behavior long before a first physical mock-up is built.

VPs are executed and analyzed in a **Virtual Environment** (VE). A VE is a VR/AR-based synthetic environment, which provides a visual, haptic, auditive, and interactive experimentation environment for the VP [86].

In our approach, we developed a methodology and technologies for simulation-based VEs for the advanced interactive analysis of self-optimizing mechatronic systems. Our VE also comprises a Virtual Test Bench (VTB) for the structured and controlled execution of experiments and tests, respectively. As such, we have advanced the idea of the classical VE [86] towards a simulation-based VE like in the Extensible Modeling and Simulation Framework (XMSF) [32] or in the High Level Architecture (HLA) [39]. That means, our approach is not limited to visualize VP models, rather than also integrates behavioral simulation models and physical hardware (Hardware-in-the-Loop) for real-time user interaction with VPs for realistic product development, analysis, and testing. For this, we have already introduced a common VE infrastructure for semi-automatic multi-domain integration of the VP [17, 172].

**Fig. 5.78** Schematic representation of the term Virtual Prototype (VP)

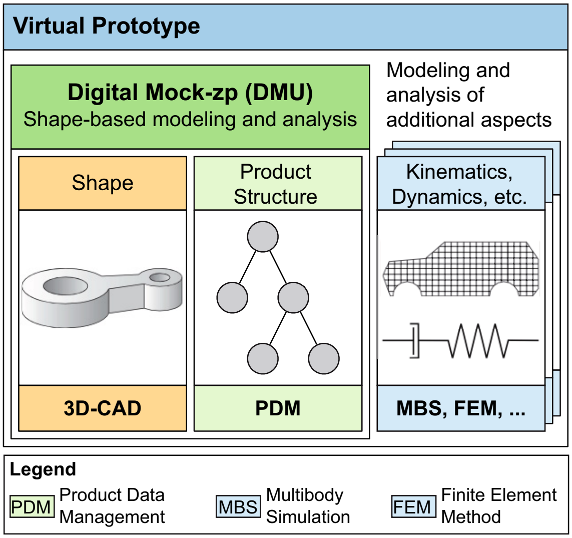

Virtual Prototype
Digital Mock-zp (DMU)
Shape-based modeling and analysis
Modeling and analysis of additional aspects
Shape
Product Structure
Kinematics, Dynamics, etc.
3D-CAD
PDM
MBS, FEM, ...
Legend
PDM Product Data Management
MBS Multibody Simulation
FEM Finite Element Method

Figure 5.79 presents an overview of the principle components of our VE and their interaction by the example of our BeBot robot [99]. The VE on top is composed of interacting Virtual Prototypes (VPs) and a **Virtual Test Bench** (VTB). The latter covers the objects of the test environment including the behavior for interacting with the VPs, i.e. stimuli, as well as a test strategy given by a verification plan for the controlled execution of experiments and tests, respectively. We can see that each VP has different aspects: shape, structure, and behavior such as kinematics, dynamics behavior, which can be given by an executable component, such as Hardware-in-the-Loop, or a domain-specific simulation model, such as a MATLAB/Simulink model, as illustrated at the bottom of Fig. 5.79.

Typically, a VE is created manually like the maritime combat simulation in [85] or the virtual factory in [186]. That means, either interactions between the different VPs and the VE are implemented manually or by means of predefined data structure with a fixed set of variables for each VP. As of today, with the increasing intelligence of mechatronic systems, the number of considered system components and their interaction significantly increases. The increasing complexity of data structures and their interaction both make the manual integration of VPs to a simulation-based real-time VE infrastructure highly time-consuming and error-prone. Therefore, we have developed an agent-based approach where software agents [108] identify the physical and non-physical interaction between single VPs automatically and link them to a VE. An agent compares two function structures given as standard models of the product development process and identifies similarities for automatic model linking.

Before we will introduce our approach for VTB automation, the next two sections outline our concepts for automatic model linking and their linking to the visual representation.

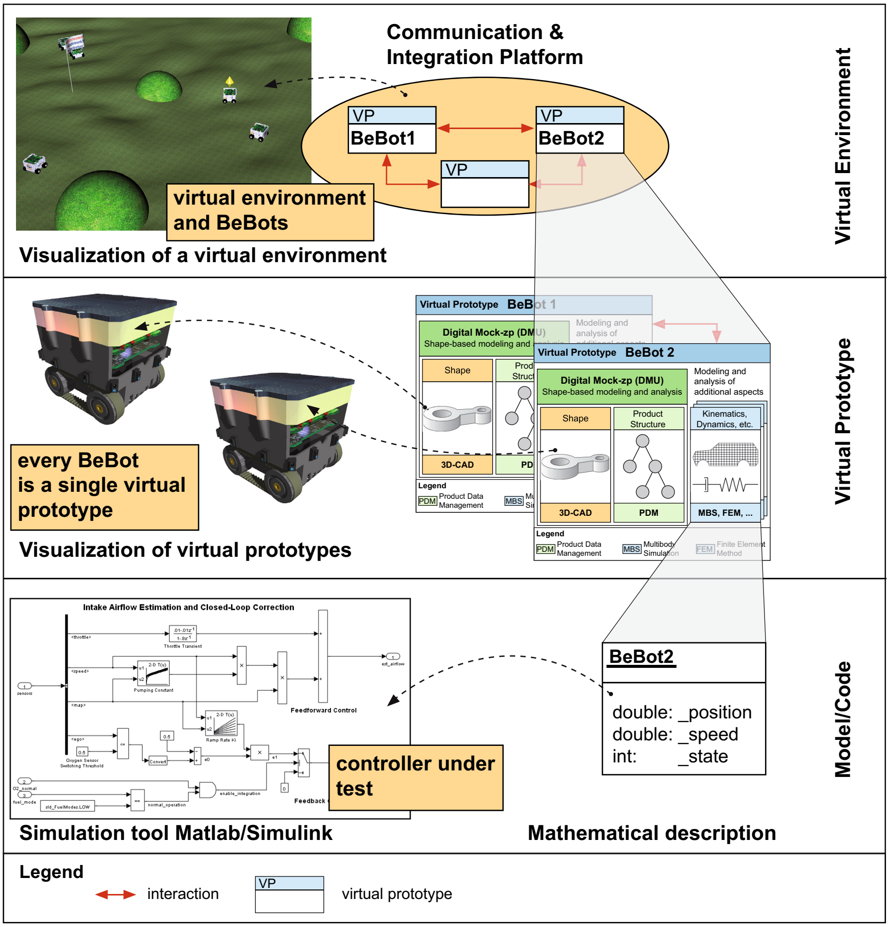

Communication & Integration Platform
VP BeBot1
VP BeBot2
VP
virtual environment and BeBots
Visualization of a virtual environment
Virtual Environment
every BeBot is a single virtual prototype
Visualization of virtual prototypes
Virtual Prototype BeBot 1
Virtual Prototype BeBot 2
Digital Mock-zp (DMU)
Shape-based modeling and analysis
Modeling and analysis of additional aspects
Shape
Product Structure
Kinematics, Dynamics, etc.
3D-CAD
PDM
MBS, FEM, ...
Legend
PDM Product Data Management
MBS Multibody Simulation
FEM Finite Element Method
Virtual Prototype
Intake Airflow Estimation and Closed-Loop Correction
Feedforward Control
Feedback
controller under test
Simulation tool Matlab/Simulink
BeBot2
double: _position
double: _speed
int: _state
Mathematical description
Model/Code
Legend
interaction
VP virtual prototype

**Fig. 5.79** Schematic overview of the composition of a virtual environment

## 5.6.2 *Automatic Model Linking*

Rafael Radkowski

Our automatic model linking is based on Semantic Web technologies. The Semantic Web (SW) facilitates machines to capture the content of web pages and other similar documents [22]. Thus, machines can automatically link information from different sources. The **Resource Description Framework** (RDF) plays a decisive role for the SW. RDF is a description language, which is used to annotate the content of a web page; it is the syntax for meta-data of a web page. The underlying model is based on a directed graph. The nodes of the graph denote resources, while the edges denote properties. The idea of RDF is to describe complex facts by a network of simple RDF statements. A RDF statement consists of a subject, a predicate, and an object. The predicate is the most important part of the semantic. It is defined as a W3C (World Wide Web Consortium)-standardized predicate for the description of business cards. The SW can only function successfully if all participants have the same understanding of these predicates and interprets them in the same manner.

A reasoning system is necessary to identify relations between two RDF-annotated web pages. RDF represents the database only. For that purpose, query languages are used to query the necessary information. Queries need to be transformed to a form where reasoning is possible by processing production rules.

Some researchers have already used RDF and the related reasoning mechanism for the engineering of technical systems. For instance, Bludau and Welp (2012) [27] have developed a framework, which supports engineers during the development of mechatronic systems. Their framework searches for active principles and solution elements, which meet a given specification. Restrepo (2007) [177] uses SW techniques to search for design solutions for a given problem. He has developed a database, which contains different design solutions; Simulink RDF annotates every solution. A reasoning mechanism searches for solutions for the given design problem. Ding et al. (2009) use XML-based annotations to annotate CAD models with design constrains, goals, relationships, and bounds [50]. They mainly annotate geometric, topological, and kinematic properties of a given design. Their approach can be utilized to find an optimal design solution during the product development process. The authors use XML as a notation basis, but their notation is similar to a RDF notation. Ding et al. (2009) developed an XML-based product representation that also allows an annotation of geometrical properties [49]. For further information, Li et al. (2009) present a classification of different annotation approaches [135]. They all demonstrate the importance of software agents and annotation techniques in engineering design, on which our approach is based.

The main principle of our approach for agent-based automatic model linking is outlined in Fig. 5.80. On the bottom left of the figure, we can see the example of two Virtual Prototypes (VP), which are linked to a Virtual Environment (VE). A software agent represents each VP.

In this example, each VP includes two models: a 3D model and a behavioral model. Both models are shown at the top of the figure with the 3D model on the left and the behavioral model on the right. The latter is illustrated by a MATLAB/Simulink screenshot. The application contains and processes a model that simulates the behavior. Both models are annotated. Therefore, an RDF-notation is utilized; the annotations describe the purpose of the models. Normally, more than two aspect models (3D model and behavior) and one VP are used.

The main task of the software agent is to combine both aspect models (3D model and behavior) to one VP and to integrate them into the VP-template, which is provided by the VE. As shown in Fig. 5.80, this requires five steps.

The first step is an initialization by a user (1). Normally, the user specifies one model (3D model or behavior model) as the origin. The objective of the agent is to identify the other models and to integrate them into the template of the VP. Therefore, the agent searches for every available model. A service directory of the agent platform references them. The annotation of every available model is read (2). The

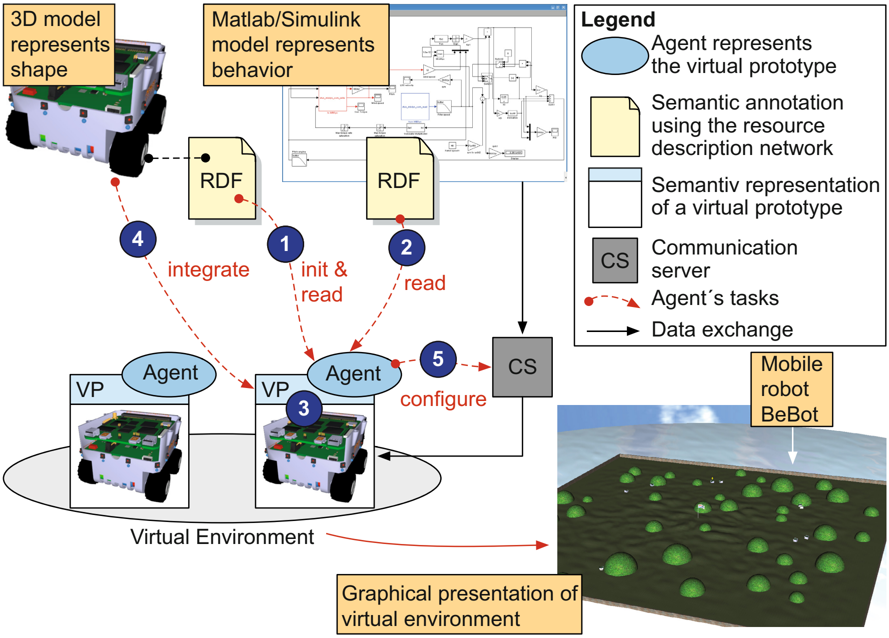

3D model represents shape
Matlab/Simulink model represents behavior
Legend
Agent represents the virtual prototype
Semantic annotation using the resource description network
Semantiv representation of a virtual prototype
CS
Communication server
Agent´s tasks
Data exchange
RDF
RDF
4
integrate
1
init & read
2
read
VP
Agent
VP
Agent
3
5
configure
CS
Mobile robot BeBot
Virtual Environment
Graphical presentation of virtual environment

**Fig. 5.80** Automatic model linking overview

agent compares the RDF model of the 3D model with the RDF model of the behavior model (3). A set of production rules is used for this task. If two models pass the production rules, the agent assumes them to be similar.

Next, the 3D model is integrated into the VE by loading the model and including it into an internal data model (4). However, the behavioral model cannot be included by simply importing it. Since the processing of the behavioral model is very resource consuming, it is executed on a separate computer system. As only simulation results are required for an analysis of the VP, only the results are transmitted by means of a communication server (CS) to the main host. The CS manages the communication between the simulation software and the VP/VE. The agent configures the CS and establishes the communication between the behavior model and the 3D model (5).

### 5.6.2.1 Semantic Annotations with the Resource Description Framework

After automatic linking, the VP models are enhanced by semantic annotations by means of the RDF (Resource Description Framework) language [22]. RDF provides a syntax for web page meta data, where the underlying model is based on a directed graph. The nodes of the graph refer to resources, the edges to properties. The idea of RDF is to describe complex facts by a network of simple RDF statements. We apply RDF as an annotation language to describe the context of each VP model. The challenge of the annotation is to identify the relevant elements of a specific model, which are required to conduct the automatic integration of the model. At this

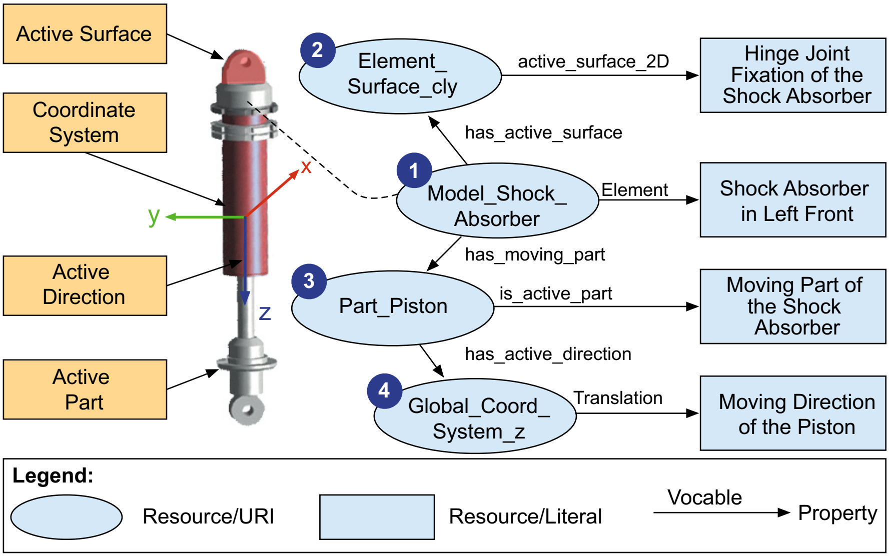


**Fig. 5.81** Example of the semantic annotation of a 3D model

step, we presume the availability of VP models of the following aspects: shape (3D model), behavior, functions, and activations.

For semantic annotation, we will outline the main concepts by the example of a shock absorber as given in Fig. 5.81. Four items of a 3D model are annotated: the entire part, the active surfaces, the subparts, and the active directions:

**Entire part** *(1)*: The resource is linked to the variable, which represents the model, normally a file. In this example, the name of the model is *Model_Shock_Absorber*. The variable is annotated by the predicate *element*. To describe the *element*, a literal is used. In this example it is ‘Shock Absorber in Left Front ‘.

**Active surface** *(2)*: The active surfaces of a component are the surfaces, which fulfill the functions of this component [158]. The resource is linked to the variable in the data structure of the 3D model, which represents the active surface. In the example, the name of the variable is *Element_Surface_cly*. *active_surface_2D* is the predicate, which defines the item as an active surface.

**Active part** *(3)*: This type of part moves to cause an effect of the entire model. The resource refers the variable, which describes the main part in the data structure of a 3D model; in this case it is the entire piston. The word *has_moving_part* is used as RDF predicate to annotate to the subpart, which describes the part in the structure of the entire 3D model; the variable's name is *Part_Piston*. Furthermore, to describe this active part, the predicate *is_active_part* is used. It facilitates the annotation with a literal. In this case the literal is: ‘Moving Part of the Shock Absorber‘.

**Active direction** *(4)*: The fourth annotation type is the active direction. According to Pahl and Beitz (2007) [158], the active direction describes the direction,

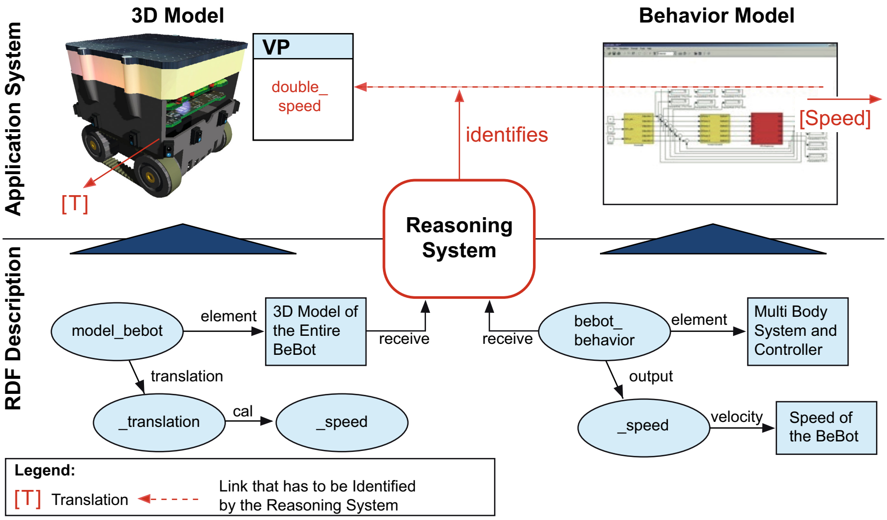


**Fig. 5.82** Schematic overview of the reasoning using an example

into which a function of a component effects. In Fig. 5.81, the piston of the shock absorber is the active part, which active direction should be described. The variable *Global_Coord_System_z* describes the coordinate system; it describes the direction of moving. It is attached to the resource *part_piston* by the predicate *has_active_direction*. In addition, the resource *Global_Coord_System_z* needs to be annotated with a human understandable literal. In the example, the literal is ‘Moving direction of the Piston’. It is attached to the resource by the predicate translation.

However, though we outlined our concepts by just one example it should be sufficient to give an impression to show how the semantic annotation with RDF is applied, and which elements of a 3D model are necessary in order to describe the purpose and functionality of a 3D model in a natural way (literals). In total, we have defined 36 RDF keywords to describe active surfaces and directions as well as the parts and subparts of an assembly. Further details can be found in [171].

#### 5.6.2.2 Software Agent Reasoning

Recall here that the software agent has two major tasks. First, it has to identify similar aspect models and, second, it has to establish the communication and the exchange of data between different software tools. The following paragraphs will outline the reasoning mechanism for establishing the communication infrastructure as sketched by the example in Fig. 5.82.

Figure 5.82 shows a 3D model of our miniature robot BeBot on the left side and a behavioral (MATLAB/Simulink) model of the robot on the right. We presume that both models are already annotated by RDF, where the example just shows a small portion of it, the variable speed. As a BeBot can fulfill different tasks in a team, the variables of the behavior model need to be linked with the related variables of the 3D model and with other VPs. For that, a reasoning mechanism identifies variables that are related to each other. In general the software agent compares the RDF models, two at the same time, and converts the results of the comparison into a numerical value. This numerical value expresses the similarity of two models, respectively their variables. The comparison is based on production rules. Each production rule has the form:

$$IF\,(Condition\,C_1\,\&\,Condition\,B_1\,\&\ldots\&\,Condition\,C_n\,\&\,Condition\,B_m)$$
$$THEN\,A_1;\ldots;A_0$$

Conditions of type $C$ are predicates of the 3D model, conditions of type $B$ are predicates of the behavior model. By applying these production rules a set of corresponding predicates is identified. As corresponding predicates each pair of predicates is defined, which describes the same meaning of an item. For instance, condition $C$ states *translation & cal* (calculated) and condition $B$ states *output & velocity* are defined as corresponding predicates; they result in an output $A = 1$. Otherwise they result in an output $A = 0$. The result of this calculation is weighted by a weight value $g$:

$$A_0 = Ag + E$$

The value g indicates the importance of a production rule. The term $E$ is an offset. It is calculated by comparing the literals of each pair of corresponding predicates. This is done by a statistical phrase analysis (see also [171]). A vector describes the results of every production rule:

$$R_{similar} = A_1, A_2, \ldots, A_0$$

That vector is a rating scale for the quality of the similarity of a certain task. After the vector is determined, the agent ranges all results $R_{similar,i}$, where the index $i$ refers to a certain production rule of two compared models. A statistical method is used for this comparison, the so-called squared ranking. This method calculates a likelihood value $p(i)$ for each corresponding pair of predicates:

$$p(j) = \frac{1}{size} \cdot \left(E_{max} - (E_{max} - E_{min}) \cdot \frac{(R_{similar,j} - 1)^2}{size - 1}\right)$$

with two rating values $E_{max}$ and $E_{min}$. These values express the estimated amount of minimal and maximal corresponding predicates, respectively the number of possible relations. The equation assigns a numerical value to each production rule and expresses the fulfilled rules by a numerical value. A high value indicates the similarity of the compared variables. The agent links all data, which value $p(j)$ exceed a threshold:

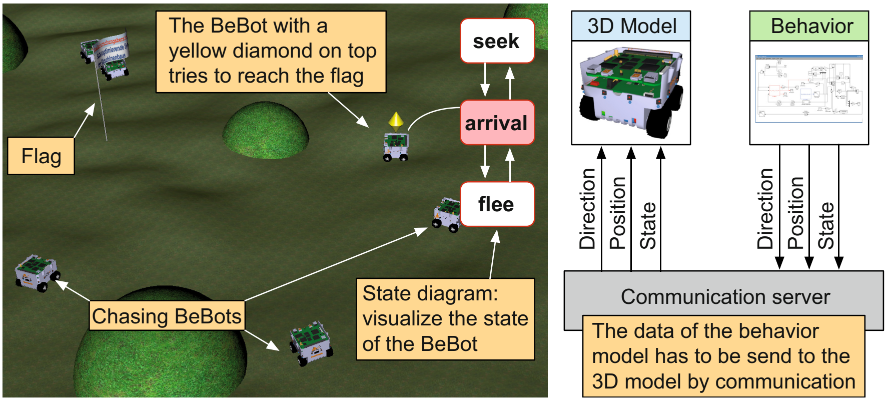

The BeBot with a yellow diamond on top tries to reach the flag
seek
arrival
flee
Flag
Chasing BeBots
State diagram: visualize the state of the BeBot
3D Model
Behavior
Direction
Position
State
Direction
Position
State
Communication server
The data of the behavior model has to be send to the 3D model by communication

**Fig. 5.83** Overview of the application (left), the architecture of the application (right)

$$p(i) > p_{threshold}$$

The value p(i) needs to cross a threshold $p_{threshold}$. At this time the threshold is determine empirically. After this decision, the agent establishes the communication between the behavior model and the 3D model. Further information about the communication infrastructure and the behavior of the agent inside the VE has been presented in [173].

### 5.6.2.3 Application Example

To proof our concepts of automatic model linking, a software prototype was developed and validated by the BeBot robot application example [99]. Figure 5.83 shows a screenshot with an overview of the VE on the left. In the environment, the Flag is located in the middle of the environment and spheres are placed as obstacles for the BeBots around it. The BeBot with the diamond on top tries to capture the flag. On the right, it shows the corresponding infrastructure of the VPs.

Each BeBot is represented by a 3D model and a behavioral model. Both models are annotated by RDF [171]. The annotation of the 3D model describes the input variables to set the position and direction of a robot as well as a state diagram to visualize its current state. The behavioral model provides the position and direction of each BeBot. Both applications (VE and behavior) need to be linked by the agent, the agent has to identify the variables and link them respectively.

In summary, the agent is able to realize the communication between both models/applications utilizing the RDF-based annotations of both models. The desired application can be realized, without any need for a user to describe the communication manually.

## *5.6.3 Visualization Agents*

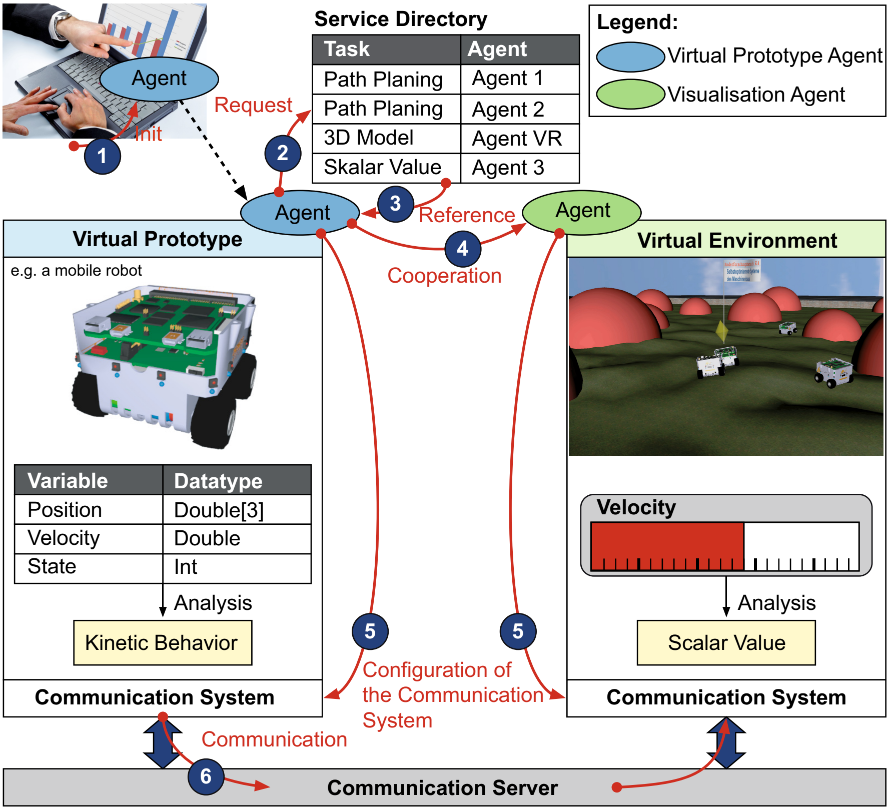
Rafael Radkowski

In engineering, software agents are utilized in many different application fields. Agents are mainly used to support the design process by making decisions, which are based on a large amount of data. Mendez et al. (2005) describe an agent-based software architecture for agents in virtual environments [142]. They introduce the concept of expert agents. Expert agents are software agents with an expert knowledge in a specific technical domain. Based on this knowledge, the expert agent is capable of finding a solution to solve a specific problem. The paper introduces a similar idea. However, their desired tasks are training tasks. Galea et al. (2009) present a framework for an intelligent design tool that assists a designer, while working on micro-scale components [67]. They do not label their framework as software agent, but they use a similar artificial intelligence technique to model the knowledge and the reasoning system. Multi-agent systems have also been used to support engineers in time-critical tasks [161]. An agent aggregates relevant information from other agents that represent different members of an engineering team. Thus, an engineer gets the right information at the right time. Baolu et al. (2009) propose the so-called Multi-Agent Cooperative Model (MACM) [15]. It is a product design system that facilitates easy access to similar data of different products. The system facilitates the product design and manages product data. With its aid the product design cycle will be shortened. Geiger et al. (1998) introduce the agent modeling language SAM (Solid Agents in Motion), a language to describe 3D models in virtual environments and their behavior [71]. In contrast to our work, SAM covers the complete visualization of animated processing of SAM-specific rules rather than links to arbitrary behavioral models.

In the following, we will describe the concept of visualization agents for linking visual representations to VP models. For this, we presume two different agents: one agent for the VP (VP-agent) and a second one for the visualization (Vis-agent). We also presume an agent platform, which is formed by a set of interacting agents, which finally form the VE. Each agent contains an internal data model. This data model describes the represented object like meta-data. Along the lines of the previously introduced linking of models, we apply RDF in combination with a reasoning mechanism to identify similarities of annotations between visualization and model agents. As a result, if models are identified as similar, we assume that the visualization is suitable to explain the data of a VP. In the following, an overview of the entire concept is presented. Then the necessary agent models are described and finally, the reasoning mechanism is introduced.

### 5.6.3.1 Concept

Figure 5.84 shows a schematic overview of the basic principles. On the left side, a box represents the VP of a mobile BeBot robot. The box on the right side indicates



| Task | Agent |
|---|---|
| Path Planing | Agent 1 |
| Path Planing | Agent 2 |
| 3D Model | Agent VR |
| Skalar Value | Agent 3 |

| Variable | Datatype |
|---|---|
| Position | Double[3] |
| Velocity | Double |
| State | Int |

**Fig. 5.84** Concept of the visualization agents

a VE with visualizations. Software agents are associated to the VP (VP-agent) and the visualization (Vis-agent).

The objective of both agents is to identify an appropriate visualization by communication and cooperation. This visualization should help a user accomplish a certain task. In the following, detailed steps are explained, which are necessary to identify a suitable Vis-agent for the visualization of a specific VP-agent along the six steps of Fig. 5.84.

In step *(1)*, a user needs to initialize the VP, the simulation, and the VE. At the beginning, the agent platform is initialized and the agents start to operate simultaneously. The user needs to specify the task, which he or she wants to carry out, e. g. to analyze the kinetic movement or to inspect the parts of the VP.

In step *(2)*, the VP starts to search for an appropriate visualization for the VP and its data. For this, the VP-agent contacts a service directory provided by the VE and queries for reachable Vis-agents. It contains a list of all reachable agents, sorted by a category of tasks. The VP-agents submit a desired category, which meets the kind of visualization the VP-agent searches for. Normally, more than one visualization facilitates the visualization of the data of the VP. Thus, in step *(3)*, the VP-agent receives a list of potential Vis-agent candidates.

In step *(4)*, the VP-agent contacts each Vis-agent with the reference from the service directory. Thereby, it submits data about the functions of the VP and data about the task the user desired to apply to every Vis-agent. Each Vis-agent compares this data with two internal data models. These data models characterize the capabilities of a Vis-agent. A similarity-vector Evis is calculated. This vector and its numerical values represent the capability of the agent to visualize the queried task and data. This vector is returned as a result to the VP-agent. At the end of this step, the VP-agent has a set of similarity-vectors, one for each Vis-agent.

Next, the VP-agent compares the different similarity-vectors and by this, it compares the different Vis-agents. Therefore, a reasoning mechanism is used. After the VP-agent has decided for one Vis-agent *(4)*, they start to cooperate.

In step *(5)*, the visualization is realized. Therefore, the data of the VP needs to be submitted to the Vis-agent and its represented visualization. Figure 5.84 shows a simple example: the VP has a 'velocity' that needs to be visualized. The Vis-agent on the right side can visualize this by a bar chart. For that, the 'velocity' values need to be transmitted to the Vis-agent. To realize this data exchange, a communication server is used [173]. This communication server manages the data exchange between different connected programs. In the example, this is a program that simulates the VP and its behavior and a VE that hosts the visualization *(6)*. The task of both agents is to configure this communication server and by this, configure the data exchange. The VP-server informs the communication server about the attributes it wants to allocate. The Vis-agent informs the server, what data it requires. If the requested data is available, the data exchange starts until an agent stops its operation.

#### 5.6.3.2 Data Models

Agents maintain three different RDF-based data models to represent their knowledge: a task model and a function model for the VP-agent, and a visualization model for the Vis-agent.

Task Model

A task is defined as *'the application of methods, techniques, and tools to add value to a set of inputs – such as material and information – to produce a work product that meets fitness for use standards established by formal or informal agreement'* [204]. A common technique to specify a task is a block diagram where each block represents a certain activity and the entire diagram represents the task (cf. Fig. 5.85). A string inside the block denotes the activity, e. g. 'Check the impulse response'. Incoming arrows represent input data (objects or information), which are processed during the activity. Information can be the velocity of a mobile robot, for instance, or an object of the computer-internal representation of the shape of the VP. In addition, an activity may also refer to a method and a tool. To concretize the task, boundary conditions can be specified. For instance, this can be the required amount of data.

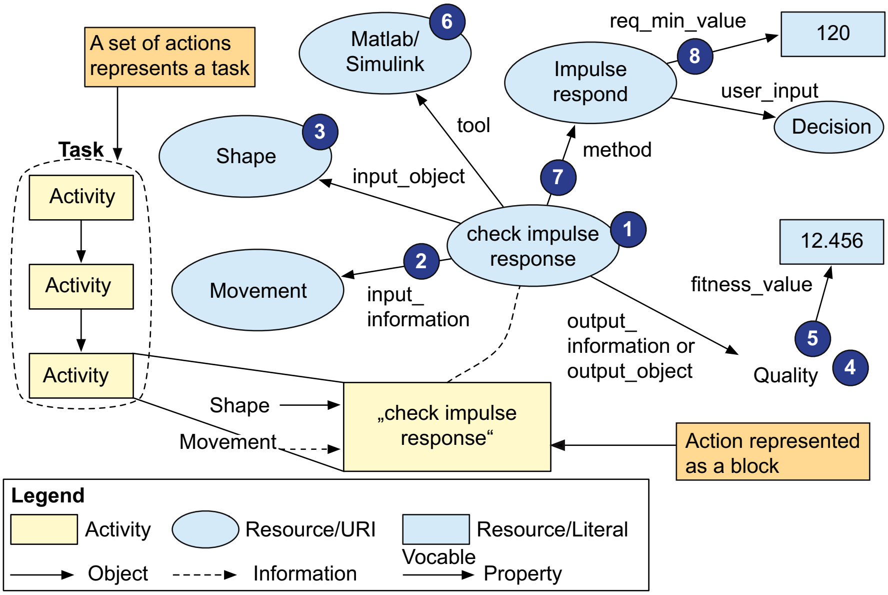


**Fig. 5.85** RDF-Description of a task model

To describe this task model as computer-internal representation, an RDF-based notation has been developed, which covers the definition of a set of resources and properties describing a task. Figure 5.85 shows an extract of the resulting RDF-based notation for one action. The following resources and properties are used in that figure:

- **Activity** *(1)*: The activity itself is the main element. It is specified by a resource, which keeps a string of the action itself.
- **Input information** *(2)*: The activity has a property *input_information* to specify the incoming information. The property refers to a resource, movement in the shown example. This resource keeps a link to the computer-internal data of this information.
- **Input object** *(3)*: The activity uses a property *input_object* to refer to the incoming objects. The property points to an additional resource, which contains a link to the computer-internal representation of this object.

The properties *output_information* and *output_object (4)* are used to refer to the outgoing information and objects. The properties refer to resources, too. Every activity can use multiple input and output objects and information.

- **Fitness value** *(5)*: Every input and output object and information uses a property *fitness_value* to express a numerical value or a set of numerical values that quantifies the objects and information. It is an optional property. It refers to a literal that contains the numerical value.

- **Tool** *(6)*: The vocab tool labels a property of the activity to describe an additional software tool. This software tool is utilized to carry out the named activity. The property refers to a resource containing a link to this certain tool. This property is optional.
- **Method** *(7)*: Every activity needs one method, which is utilized to process the activity. The vocab method is used to express this property. It refers to an additional resource. At this time, only the resource keeps a name of the method. The methods are provided in a database. The user can only select a method.
- **Conditions** *(8)*: Every method can be concretized by additional conditions. Two conditions are used. The first one is a requirement value. It denotes a minimum amount of data that is necessary to process this method. This property is expressed by the keyword *req_min_value*. It refers to a literal containing a numerical value. The value quantifies the requirement. The second condition expresses whether a user input is necessary during this activity or not. For this, the keyword *user_input* is used. It labels a property, which refers to a resource. This resource contains a statement that expresses the type of user input. For instance a Boolean decision (yes/no). The conditions are optional properties.

Function Model

Figure 5.86 shows a schematic overview of the function model. It is defined to specify the functionality of a VP in respective to the product under development. Therefore, a function structure according to Pahl/Beitz (2007) is used [158]. For the graphical presentation of the function structure a block diagram is a common technique. Each block represents a function. A function is defined as 'Operation, activity, process, or action performed by a system element to achieve a specific objective within a prescribed set of performance limits'. According to Pahl/Beitz (2007) it is expressed by a substantive and a verb [158]. The substantive names the object that is processed by the function. The verb names the process or the activity the technical system carries out. To build up a function structure, the functions of a technical system are connected by the flow of material, energy, and information. The arrows in Fig. 5.86 show these flows. The entire function structure represents a model of the functionality of the technical system.

To use the function structure as a knowledge model for an agent, a formal computer internal representation has been developed. Therefore, we have developed a RDF notation, too. Figure 5.86 shows an extract of the developed RDF scheme in order to introduce the resources and properties and to demonstrate its application. The example explains how a function model can be built up and which properties are necessary to describe the functionality of a VP by RDF. The following notation is used:

- **Function** *(1)*: The function itself is expressed by a resource. The resource keeps a character string of the function. It is the main resource of every function and it is required.
- **Function term** *(2)*: To facilitate an automatic processing of the function term, the function uses a property *function_term*. This refers to an empty resource that

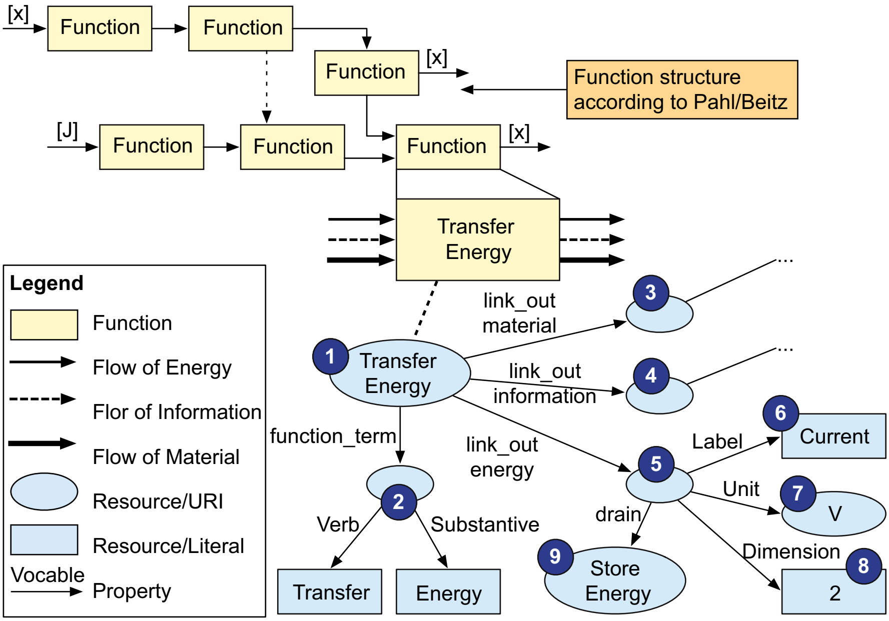


**Fig. 5.86** Schematic presentation of the RDF notation of a function structure

points to the substantive and the verb of the function. The property substantive refers to a literal of the substantive. The property verb refers to the function verb literal.

- **Flow of energy, information, and material**: To model these three types of flow, the function uses the properties *link_x_material (3)*, *link_x_information (4)*, and *link_x_energy (5)*, where x is a wild-card for in or out. The property refers to an empty resource.
- **Attributes**: The flow of energy, information, and material need to be specified by three additional properties. These properties are the label, the unit of the technical dimension, and the dimension of the value. The property label *(6)* refers to a literal, it contains a character string that names the flow. The property unit *(7)* points to a resource. This resource keeps a value of a technical dimension; in the example the unit 'V' for voltage is shown. The last property depicts the dimension of the flow. A scalar, a vector, or an array can model the flow. For this, the property dimension *(8)* is used. It refers to a literal to characterize the dimension.
- **Source and drain** *(9)*: Every flow has a source and a drain. To specify them, the properties source and drain are applied. Both properties refer to a resource that contains a link to the related function.

Visualization Model

A visualization is defined as a technique to create images, diagrams, 3D models, and animations to communicate and to explain abstract data. For instance, it can be a bar

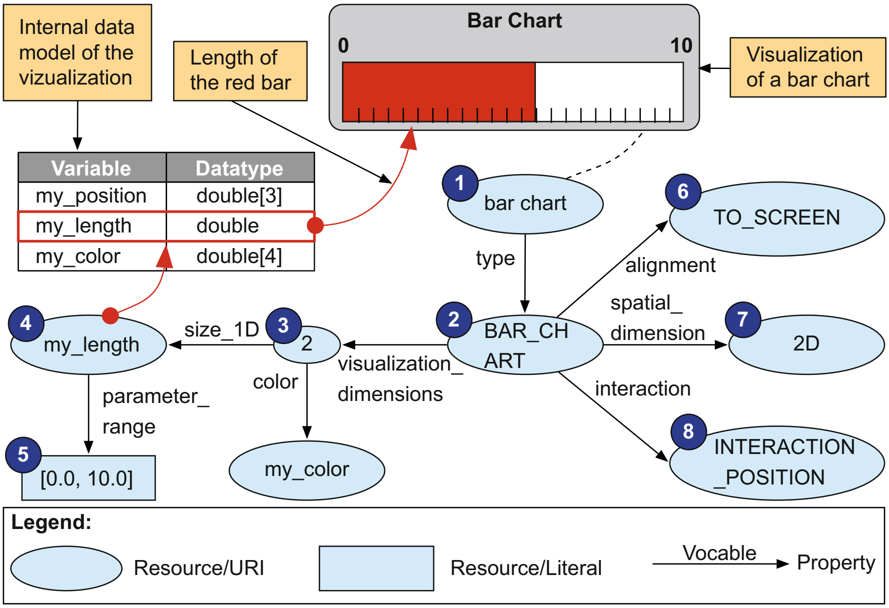


**Fig. 5.87** Schematic representation of the visualization model

chart as a visual representation of a scalar value (cf. Fig. 5.87). In the context of the visualization agent, visualizations are diagrams and 3D models. Both of them are a part of the VE. A visualization is annotated by an RDF-notation, too. Figure 5.87 shows an overview of the used resources and properties and how they are applied. As an example, a bar chart is used. To define the RDF-notation, it was necessary to identify elements and attributes that specify a visualization and its capabilities. The following resources and properties are used:

- **Visualization** *(1)*: The visualization itself is modeled as a resource. The entry of this resource refers to the internal data model of the visualization. This key element is required.
- **Visualization type** *(2)*: To specify the type of visualization the related resource has a property *type*. This property refers to a resource that denotes the visualization by a keyword. In the example, the keyword BAR_CHART specifies a bar chart. Other keywords are SYMBOL, ICONS, NET, TREE, and some more. Each of them represents a certain type of visualization.
- **Dimensions** *(3)*: Every visualization has a set of visual variables. These visual variables are modified to express abstract data by a graphical representation. The property *visualization_dimensions* specifies the number of visual variables each visualization provides. It refers to a resource that contains the number of modifiable visual variables.
- **Visual variable** *(4)*: This property is used to specify the visual variables itself. To describe them, visual variables according to Bertin (1983) [24] are used. These

variables define the size of a visualization, the position, the orientation, the grey scale value, the color, the texture, and the shape. They are transferred to properties like *size_1D*, *size_2D*, *size_3D*, *position*, *orientation*, *color*, etc. For instance *size_1D* specify a visualization, which size can be modified in one dimension. In the example of the bar chart, it is the length of the bar. The property refers to a resource. This resource keeps a link to a variable of the visual variable, which represents its length inside the computer-internal data model. In the example shown, it refers to the double *my_length*.

- **Parameters** *(5)*: To concretize the visualization, the visual variable can be limited by a set of parameters. At this time, two parameters respectively properties are used: *range* and *threshold*. The property *range* specifies the boundaries of a dimension. For instance, the bar of the bar chart is limited by a minimum and a maximum value. In the example, it ranges from 0 to 10. The property *threshold* names a threshold, which is shown by the visual variable.
- **Alignment** *(6)*: The property *alignment* specifies the spatial alignment of the visualization. It refers to a resource that contains a keyword. Used alignments are HUD (head-up display), TO_SCREEN, TO_MODEL, and some more. For instance, TO_SCREEN means that the visualization is rotated into the viewing direction of the user automatically. Thus, the user sees the right face of the visualization every time.
- **Spatial Dimension** *(7)*: A visualization can be distinguished by its spatial dimension. This feature is specified by the term *spatial_dimension*. The property refers to an additional resource, it contains the dimension: 0D (Points), 1D (Lines), 2D (Surfaces), 3D (Volumes).
- **Interaction** *(8)*: The property *interaction* needs to be specified if input data from the user is necessary or possible, e.g. when a visualization should be moved on screen or the range of a bar needs to be adapted interactively. The property refers to a resource that contains the keyword INTERACTION_x, where x is a wild card for RANGE, POSITION, and some more. For instance, INTERACTION_RANGE means that the user can modify the boundaries of a visual attribute.

This data is sufficient to specify a visualization with a set of annotations. Its computer-internal representation has been integrated into an agent model to specify the visualization.

### Reasoning Mechanism

The reasoning mechanism identifies the Vis-agent, which associated visualization is adequate to visualize the data of the VP or the VP itself. In general, the reasoning mechanism compares the models and converts the results to a numerical value. This numerical value expresses the capability of a Vis-agent to visualize the data of a VP.

We apply three steps to identify a proper visualization. The first step is processed by the Vis-agent. The second and the third step are processed by the VP-agent. At the beginning, we presume that the VP-agent has submitted its models to the Vis-agent.

In the first step, production rules are used to determine the similarity between different models. A Vis-agent keeps a set of production rules to evaluate the request. Each production has the form

$$IF\,(Condition\,C_1\,\&\,Condition\,B_1\,\&\ldots\&\,Condition\,C_n\,\&\,Condition\,B_m)$$
$$THEN\,A_1;\,\ldots\,;A_0$$

Conditions of type $C_n$ are related to the function model and the task model of the VP-agent. Conditions of type $B_m$ are related to the visualization model and task model of the Vis-agent. Each visualization agent contains a set of production rules. These rules compare the referred models and determine, whether the Vis-agent fulfills the requirements of the VP-agent. If the capabilities meet the requirements, action $A_0$ is processed. Each action is an equation of the form

$$A_0 = Ag + E$$

with the term $a = 1$ if the production rule is passed and $a = 0$ if the production rule fails. The value $g$ is a weight that indicates how important the production rule is. The term $E$ is an offset; it represents the experience of the agent and describes, how useful this action was during previous uses. The value $A_0$ represents the result. The results of every production rule are combined in one vector:

$$E_{Vis} = A_1, A_2, \ldots, A_0$$

This vector is a rating scale for the quality of the visualization in a certain task. Every Vis-agent calculates this vector and returns it to the VP-agent.

In the second step the VP-agent compares all results $E_{Vis,i}$, where the index $i$ refers to a certain Vis-agent. A statistical method is used for this comparison, the so-called linear ranking. This method calculates a likelihood value $p(i)$ for each visualization:

$$p(i) = \sum_{j=0}^{m} E_{max} - (E_{max} - E_{min}) \cdot \frac{(E_{Vis,j} - 1)^2}{size - 1}$$

with rating values $E_{max}$ and $E_{min}$, which determine the estimated amount of minimal and maximal fulfilled production rules. During the development of a VP-agent, it needs to be estimated how many production rules need to be fulfilled in order to identify a suitable visualization. This estimation needs to be evaluated by the developer of a individual visualization. The equation assigns a numerical value to each production rule and expresses the fulfilled rules by a numerical value. A high value indicates that the Vis-agent is adequate to visualize the VP and the generated data of the VP.

In the third step, the VP-agent decides, which visualization agent is applied: the VP-agent takes the Vis-agent with the highest value $p(i)$. One constraint is the equation:

$$p(i) > p_{threshold}$$

The value $p(i)$ needs to cross a threshold $p_{threshold}$. At this time, the threshold is determined empirically.

The concepts of visualization agent have been implemented and proven by the following application example.

#### 5.6.3.3 Application Example

To test the concept of visualization agents and the developed models, a software prototype has been developed and a BeBot robot [99] application example has been implemented.

The software prototype has four components. The first component, a VE, is based on OpenSceneGraph[15], an open source scene graph library for the development of 3D graphic applications. The second component is a simulation for mobile robots based on Open Steer [178], an open source software library, which covers a robot model and a set of functions like seek, evade, path following, and leader following. The third component is JADE (Java Agent DEvelopment Framework). JADE is a software framework that facilitates the implementation of multi-agent systems by means of a middleware that complies with the FIPA (Foundation for Intelligent Physical Agents) specifications, a standard specification for software agents. Furthermore, it provides a set of tools that supports the debugging and deployment phases of agents. The described agent behavior has been implemented using the JADE framework. The fourth component is a communication server. It realizes the exchange of data between the three components, mentioned before. The entire system works in real time. The technical details of the server are described in [173].

In addition to the four components, the software SchemaAgent from Altova[16] is used to annotate the models. It provides a graphical user interfaces to model the resources, properties, and the entire RDF graph. The RDF model is stored in an XML notation. Finally, the software library Jena is used to implement the RDF vocabulary for the annotation, the RDF queries, and the reasoning system.[17] Jena is a framework for building Semantic Web applications. It provides a programmatic environment for RDF and RDF-Schemas including a rule-based inference engine. The inference engine has been extended to realize the method, which is described in Sect. 5.6.3.2.

Based on that platform, we will outline the basic principles of visualization by means of the Capture the Flag (CtF) application example.

[15] www.openscengraph.org

[16] http://www.altova.com

[17] http://jena.sourceforge.net/

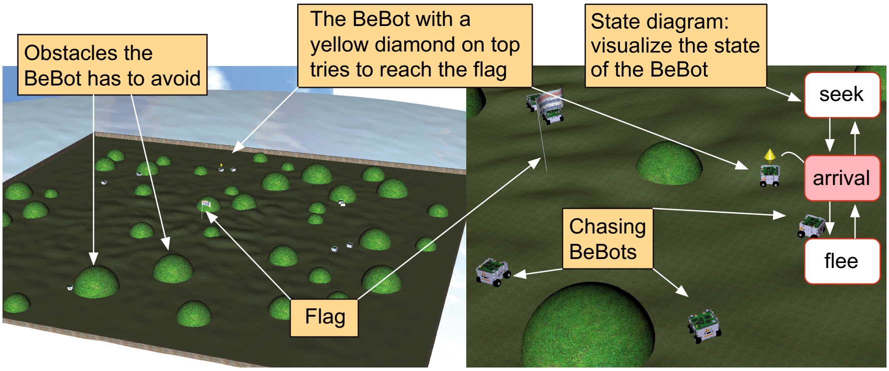

Obstacles the BeBot has to avoid
The BeBot with a yellow diamond on top tries to reach the flag
State diagram: visualize the state of the BeBot
seek
arrival
flee
Chasing BeBots
Flag

**Fig. 5.88** Overview of the virtual environment (left), detail view of the test (right)

CtF is an example, which is originally based on a game where a hunter has to capture a flag, the other players chasing the hunter and try to prevent him from capturing the flag. In our case the players are the BeBots with one hunter and n chasers. The BeBots operate autonomously without any interactions from a user. Figure 5.88 shows two screenshots from the application. The left part shows an overview of the VE.

The flag stands in the middle of the environment with spheres as additional obstacles for the BeBots. The right part shows a detailed view to the scene. The BeBot with the diamond on top identifies the hunter. A state machine with six states models the behavior of a BeBot. Each state represents a type of behavior: seek, flee, obstacle avoidance, robot avoidance, pursuit, and arrival. The BeBots decide autonomously which state is active; the decision is based on a rule system.

To test the visualization agents, the BeBots and one visualization (state diagram) have been implemented and represented by software agents. The task, the behavior, and the visualizations have been specified by the introduced RDF notation. The CtF task has been specified by a task model, the behavior by a function model, and the visualizations by a visualization model.

The task of the Vis-agents is to visualize the different states by a state diagram. Therefore, the VP-agent needs to identify the correct Vis-agent. Finally, the application has proven the correctness of our models and it was possible to identify a visualization.

### *5.6.4 Virtual Test Bench*

Wolfgang Müller and Tao Xie

The complexity of self-optimizing systems requires a systematic and thorough verification methodology in order to guarantee their adaptive run-time behavior. In the context of the VE, our test bench is based on the principles of mutation analysis, which we have extended towards a self-optimizing Virtual Test Bench (VTB) for the simulation-based analysis of self-optimizing systems.

Mutation analysis defines a unique coverage metric that assesses the quality of test cases of a test bench with respect to coding errors. It was originally introduced for software testing in the 90's [48]. Since 2007, mutation analysis was adopted for Register-Transfer Level (RTL) hardware design verification [191]. At that time, the professional mutation analysis tool Certitude(TM)d was introduced by CERTESS (now Synopsys) with the support of VHDL, Verilog, and C [89].

The remainder of this subsection first outlines the basic principles of mutation analysis before our self-optimizing test bench with a brief BeBot robot [99] application example is introduced.

### 5.6.4.1 Mutation Analysis and Simulation

Mutation testing is a fault-based simulation metric. It highlights an intrinsic requirement on simulation test data that they should be capable of stimulating potential design coding errors and propagating the erroneous behavior to check points. Mutation testing measures and enhances a simulation process as shown by Fig. 5.89.

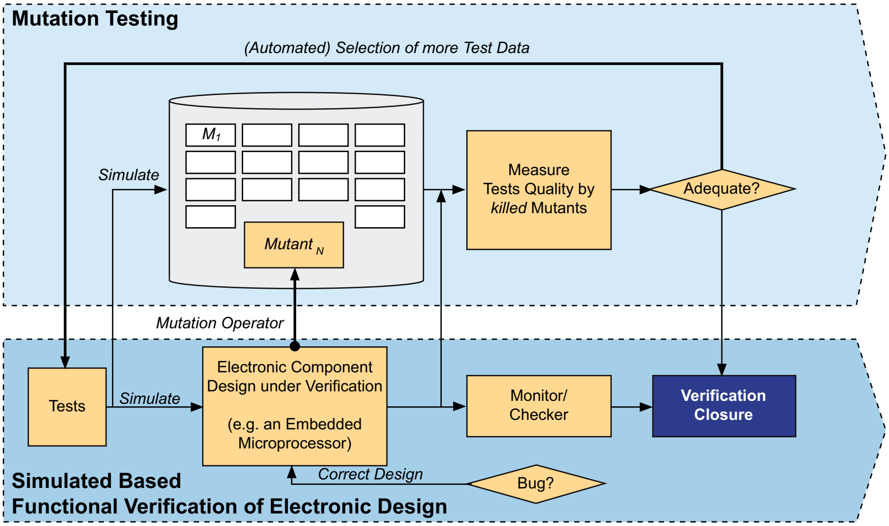

Mutation Testing
(Automated) Selection of more Test Data
$M_1$
Mutant $_N$
Simulate
Measure Tests Quality by *killed* Mutants
Adequate?
Mutation Operator
Tests
Simulate
Electronic Component Design under Verification (e.g. an Embedded Microprocessor)
Monitor/ Checker
Verification Closure
Correct Design
Bug?
Simulated Based Functional Verification of Electronic Design

**Fig. 5.89** Principle of mutation testing for the functional verification of electronic component designs

A so-called mutation is a single fault injection into a copy of the design under verification, such as this HDL statement modification:

$$\text{a} <= \text{b and c}; \xrightarrow{mutation} \text{a} <= \text{b or c};$$

The fault-injected copy is denoted as a mutant of the design. For each test case, the mutant is simulated after the simulation of the original design and both simulation results are compared. If any simulation difference appears at the design output, this test is said to be able to kill the mutant. Each type of fault injection is called a mutation operator and dozens of such operators can be defined based on the design language under consideration. By applying these pre-defined mutation operators at different locations of a design, we can obtain a huge database of mutants. The number of killed mutants becomes the mutation coverage metric and measures the overall quality and thoroughness of a simulation process.

We consider employing random simulation as a long recognized useful lightweight method to support mutation testing. However, the lightweight nature of random simulation will conflict with the inherent computation expensiveness of mutation testing. Basically, each time a random test is generated, it should be simulated against not only the original design under verification but also all the mutants that are created as the coverage points, which can be numerous. Since the test is randomly selected and relatively aimless, this amplifies the mutation testing problem. We have addressed that problem by developing an approach for a self-optimizing test bench, which is outlined in the next paragraphs.

#### 5.6.4.2 Self-optimizing Virtual Test Bench

Our self-optimizing Virtual Test Bench is based on the combination of mutation analysis with constrained random test pattern generation. Constrained random test pattern generation is a technique, which has been introduced in conjunction with the principles of functional verification and is an offline method to generate random test patterns for intervals, which are defined by constraints.

We apply constrained Markov chains to enable effective adjustment to the probability model of random simulation. An efficiency-improving heuristic is proposed to make this adjustment by utilizing two-phase mutation testing results. Such a test bench is shown in Fig. 5.90. The self-optimizing Virtual Test Bench integrates an in-loop heuristics that dynamically adapts the test probability model to a more efficient distribution for mutation coverage. As such, we finally arrived at a self-optimizing simulation-based test bench integrated into our VE that achieves higher mutation coverage for VPs under test within less simulation time.

As a prerequisite for the dynamic adjustment, we need a probability model on test sequences that provides the possibility of parameter steering. We consider that an electronic component design has a precisely defined instruction interface, such as the ISA of a microprocessor, or the communication protocol of a bus controller. For this, test inputs in a random test generator are modeled in two layers as shown in Fig. 5.90. First, a Markov chain is used to represent sequences of tests. Each node models one type of test instruction. The selection probability on edges enables us to establish the correlation between mutation analysis efficiency and a short pattern of test sequence. Second, weighted constraints are defined on the fields of an instruction. This provides the possibility for steering test patterns towards more effective areas like corner cases.

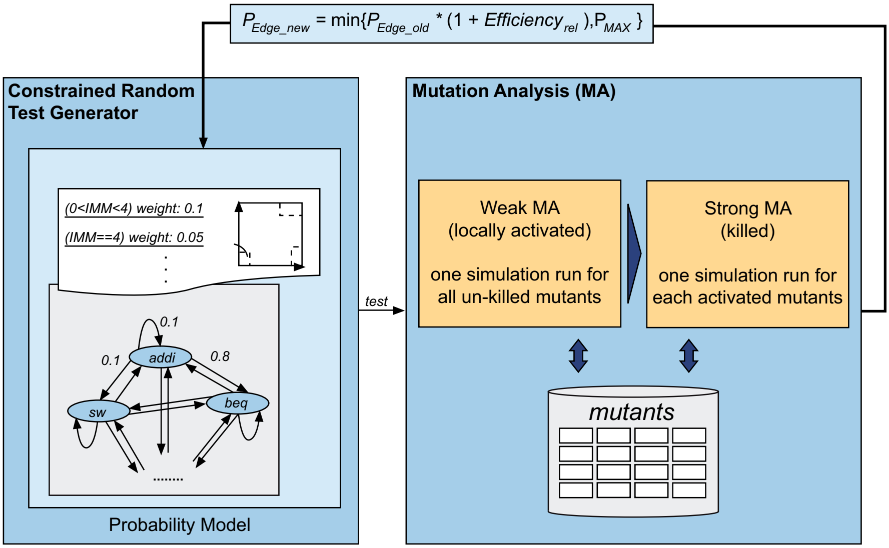

P_Edge_new = min{P_Edge_old * (1 + Efficiency_rel ),P_MAX }
Constrained Random Test Generator
(0<IMM<4) weight: 0.1
(IMM==4) weight: 0.05
0.1
0.1
0.8
addi
sw
beq
........
Probability Model
test
Mutation Analysis (MA)
Weak MA
(locally activated)
one simulation run for all un-killed mutants
Strong MA
(killed)
one simulation run for each activated mutants
mutants

**Fig. 5.90** A mutation testing directed adaptive simulation framework for the functional verification of electronic component designs

Each time a test is generated, we record the pair of Markov edge and constraint that is selected for the generation. The basic idea is to estimate the efficiency of this test on mutation analysis and use the estimation to adjust the probability of the corresponding Markov edge and constraint. This efficiency estimation should follow the unique simulation cost of mutation analysis. As the right half of Fig. 5.90 shows, we introduce at first an extra weak mutation analysis phase [104]. It uses one simulation cycle to identify the locally activated mutants. Only those are fed into a traditional, strong mutation analysis phase and fully simulated, to see, whether they are killed under the criterion that a different value appears at design output ports. Consider that $\varphi$ is the test probability distribution from a Markov-chain/constraint model, which further implies $P_{M_i\_activated}$ and $P_{M_i\_kill}$ for each mutant $M_i$ as its probabilities of being activated and killed under the current test model. On a set of $N_{Mutant}$ design mutants, this leads to an *expected simulation effort* for the mutation analysis flow in Fig. 5.90 as

$$\max_{1\leq i\leq N_{mutant}} (1/P_{M_i\_kill}) + \sum_{1\leq i\leq N_{mutant}} (P_{M_i\_activate}/P_{M_i\_kill})$$

Based on this expected simulation effort, we use the number of mutants activated by the test $N_{activated}$ and the number of its mutants killed $N_{killed}$ to estimate the efficiency of this test as

$$Efficiency = \frac{N_{killed}}{N_{activated}}$$

A low ratio means that too many mutants are merely activated and a lot of simulations are wasted in the second phase without killing the mutants. We also record this efficiency value for the last 10 tests generated and use the average $Efficiency_{average\_last\_ten}$ to derive a relative value that lies between 0 and 1.

$$Efficiency_{rel} = \frac{Efficiency}{Efficiency_{average\_last\_ten} + Efficiency}$$

By this, at the early stage of a random simulation, test patterns with high mutation kill/activation rates are encouraged. However, we observed in our experiment that in the last stage, it may well happen that no single mutant is killed in ten consecutive iterations. In such a case, the heuristic approach *changes to another mode that encourages more activation of mutants*, by first calculating *efficiency* as an adjustment value and then increasing the probability/weight of the corresponding Markov chain edge/constraint with the following value:

$$Efficiency_{rel_activation_mode} = \frac{N_{activated}}{N_{activated\_average\_last\_ten} + N_{activated}}$$

Here, it is safe for us to assume that there will always be some mutants activated. Initially, all Markov chain edges have the same probability to be selected and instruction constraints have the same weight. At the end of each iteration for test generation, the probability of the used edge, as well as the weight of the used constraint is adjusted by

$$\begin{cases} P_{Edge\_new} = \min\{P_{Edge_old} * (1 + Efficiency_{rel}), P_{MAX}\} \\ P_{Constr\_new} = \min\{W_{Constr_old} * (1 + Efficiency_{rel}), W_{MAX}\} \end{cases}$$

$P_{MAX}$ and $W_{MAX}$ are efforts to prevent the starvation of other edges/constraints, by setting an upper bound of probability to one edge/constraint. In the following example, with a model of 58 Markov edges, we set these two numbers to 0.9.

For each $Edge_i$ that flows out from the same instruction node and each $Constr_i$ on this node, we adjust their probability/weight proportionally to their old values

$$\begin{cases} P_{Edge_i\_new} = (1 - P_{Edge\_new}) * \dfrac{P_{Edge_i\_old}}{1 - P_{Edge_i\_old}} \\ P_{Constr_i\_new} = (1 - P_{Constr\_new}) * \dfrac{P_{Constr_i\_old}}{1 - P_{Constr_i\_old}} \end{cases}$$

#### 5.6.4.3 Application Example

We applied our self-optimizing Virtual Test Bench to the BeBot robots [99] in order to indicate the strength and also the current limits of the approach in the context of a Virtual Prototyping Environment. As such, we consider a path finding algorithm implemented in C as a design under test, which navigates the BeBot by means of 12 infrared sensors inside the VE of a randomly generated labyrinth. For test automation, we used an automatically generated configuration file to parameterize each simulation run, such as the terrain of the environment, starting point and target of the BeBot. A configuration generator tries to dynamically improve the test bench by utilizing results from the mutation analysis. After each run, the test bench monitored, whether the BeBot successfully finished the predefined route. The code of the path finding algorithm was mutated by the tool Certitude(TM). After applying our self-optimization heuristics, the configuration generator improves the test bench by utilizing results from the mutation analysis.

For our application example, with our original BeBot C source file as input, CERTITUDE(TM) initially generated 184 mutants by injecting various faults. All these mutants were compiled together with the VE, in the same way as the original code. Then, each of the generated184 mutants and the original BeBot code were simulated before new configurations were generated.

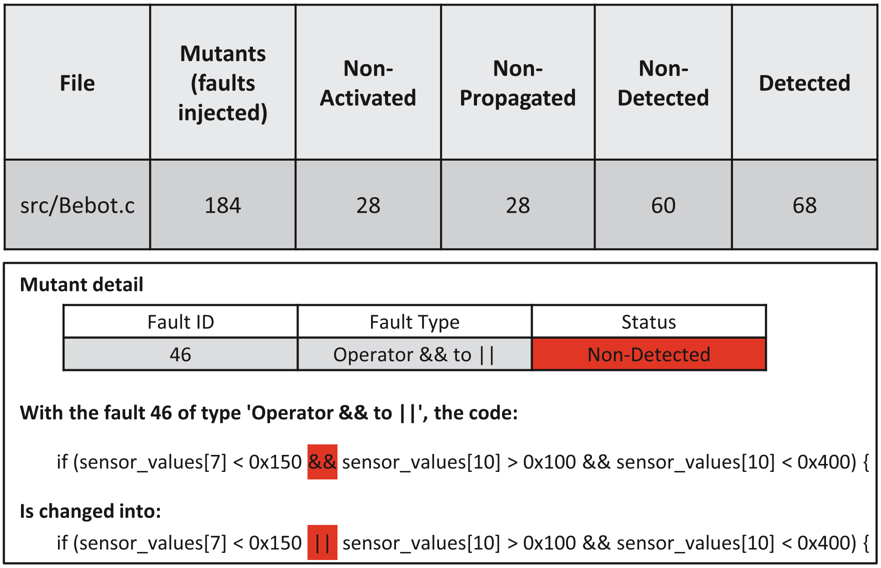
Figure 5.91 (top) shows results of the BeBot test experiments, as a summary from the Certitude(TM) report, as well as examples of mutants for the first configuration of the test environment. It shows that, at the end, the test was able to detect 68 BeBot mutants, among the total 184 mutants generated.

The remaining mutants could not be detected in this test configuration and revealed the weakness of the test patterns. These included 28 non-activated, 28 non-propagated, and 60 non-detected mutants. The status of a mutant and its injected fault is measured by Certitude(TM) as follows:

- **Non-Activated:** The fault-injected mutation statement was not executed in the simulation.
- **Non-Propagated:** The mutation statement was executed, but the execution had the same result as that from the original statement in the original design simulation.
- **Non-Detected:** The mutation statement was executed and introduced a wrong-valued behavior into the mutant simulation. However, the test bench was not able to distinguish this mutant as an incorrect design.
- **Detected:** The Test bench was able to tell that we had an error in the mutant simulation.

There were two reasons for the applied test bench not being able to detect a mutant. The first reason was that the mutant is created at a location of the code, which inherently does not induce any wrong behavior in the BeBot, like, for example, some debugging statements. The second reason was that the undetected mutant indeed reveals the weakness of our test bench. It can either be that the exercise from the test bench with the current labyrinth was not sufficient to stimulate the faulty behavior, or that the stimulated erroneous behavior did not have significant impact to be observed by the test bench.

Figure 5.91 shows at the bottom an example of such undetected mutant. The mutant with ID 46 was created by a fault injection of changing an && (logical AND) operator to || (logical OR). The Virtual Test Bench could not detect this artificial bug in the BeBot code, which indicates that an improvement of the test bench is necessary.



| File | Mutants (faults injected) | Non-Activated | Non-Propagated | Non-Detected | Detected |
|---|---|---|---|---|---|
| src/Bebot.c | 184 | 28 | 28 | 60 | 68 |

**Mutant detail**

| Fault ID | Fault Type | Status |
|---|---|---|
| 46 | Operator && to \|\| | Non-Detected |

**With the fault 46 of type 'Operator && to ||', the code:**

```
if (sensor_values[7] < 0x150 && sensor_values[10] > 0x100 && sensor_values[10] < 0x400) {
```

**Is changed into:**

```
if (sensor_values[7] < 0x150 || sensor_values[10] > 0x100 && sensor_values[10] < 0x400) {
```

**Fig. 5.91** Snapshot from CERTITUDE(TM) report for BeBot virtual test. TOP: Overall results BOTTOM: A mutant detail

Certitude(TM) is widely and successfully applied for the mutation analysis of hardware models at register-transfer level (RTL), e. g. in VHDL and Verilog, and we also successfully demonstrated our self-optimizing approach for the test bench of the MicroBlaze processor at electronic system level (ESL). In summary, our BeBot evaluations indicate that the application of Certitude(TM) also makes sense for the mutation analysis of abstract self-optimizing behavior. However, though promising, our evaluations also demonstrate there is a considerable gap between RTL and our system level applications so that further studies are still required to draw a wider conclusion.

We developed a VR- and AR-based platform for the Virtual Prototyping of self-optimized mechatronic systems with real-time user interaction. The previous section focused on the automatic configuration of VEs and on Virtual Test Bench automation. The general concepts of that framework for the integrated simulation of multi-domain VPs can be found in [17, 172]. Here, we have demonstrated the feasibility of our approach for automatic configuration VEs by means of the BeBot robot application. However, as the degree of automation is partly based on the analysis of domain-specific models, it still requires further investigation of the semantic analysis for cross-domain application and model linking.

## References

1. Adelt, P., Donoth, J., Gausemeier, J., Geisler, J., Henkler, S., Kahl, S., Klöpper, B., Krupp, A., Münch, E., Oberthür, S., Paiz, C., Porrmann, M., Radkowski, R., Romaus, C., Schmidt, A., Schulz, B., Vöcking, H., Witkowski, U., Witting, K., Znamenshchykov, O.: Selbstoptimierende Systeme des Maschinenbaus. In: Heinz Nixdorf Institut, Universität Paderborn, vol. 234. HNI-Verlagsschriftenreihe, Paderborn (2009)
2. Adelt, P., Esau, N., Hölscher, C., Kleinjohann, B., Kleinjohann, L., Krüger, M., Zimmer, D.: Hybrid Planning for Self-Optimization in Railbound Mechatronic Systems. In: Naik, G. (ed.) Intelligent Mechatronics, pp. 169–194. InTech Open Access Publisher, New York (2011)
3. Adelt, P., Esau, N., Schmidt, A.: Hybrid Planning for an Air Gap Adjustment System Using Fuzzy Models. Journal of Robotics and Mechatronics 21(5), 647–655 (2009)
4. Ali, M.I.A.H., Sitte, J., Witkowski, U.: Parallel Early Vision Algorithms for Mobile Robots. In: Proceedings of the 4th International Symposium on Autonomous Minirobots for Research and Edutainment, Buenos Aires, pp. 133–140 (2007)
5. Alur, R.: Formal Verification of Hybrid Systems. In: Proceedings of the 9th ACM International Conference on Embedded Software, Taipei, pp. 273–278. ACM, New York (2011)
6. Alur, R., Courcoubetis, C., Dill, D.: Model-Checking in Dense Real-time. Information and Computation 104, 2–34 (1993)
7. Alur, R., Courcoubetis, C., Halbwachs, N., Dill, D.L., Wong-Toi, H.: Minimization of Timed Transition Systems. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 340–354. Springer, Heidelberg (1992)
8. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
9. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Information and Computation 75(2), 87–106 (1987)
10. Antoulas, A.C., Beattie, C.A., Gugercin, S.: Interpolatory Model Reduction of Large-Scale Dynamical Systems. In: Mohammadpour, J., Grigoriadis, K.M. (eds.) Efficient Modeling and Control of Large-Scale Systems, pp. 3–58. Springer, Heidelberg (2010)
11. Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Vision-based Reinforcement Learning for Purposive Behavior Acquisition. In: Proceedings of the IEEE International Conference on Robotics and Automation, Nagoya, pp. 146–153 (1995)
12. Babitski, G.: Inferenzalgorithmen zur Auswahl ontologiebasierter Situationsbeschreibungen für ein kontextadaptives Dialogsystem. Ph.D. thesis, Technische Universität Darmstadt (2004)
13. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
14. Baldin, D., Kerstan, T.: Proteus, a Hybrid Virtualization Platform for Embedded Systems. In: Rettberg, A., Zanella, M.C., Amann, M., Keckeisen, M., Rammig, F.J. (eds.) IESS 2009. IFIP AICT, vol. 310, pp. 185–194. Springer, Heidelberg (2009)
15. Baolu, G., Shibo, X., Meili, C.: Research and Application of a Product Cooperative Design System Based on Multi-Agent. In: Proceedings of the 3rd International Symposium on Intelligent Information Technology Application, Nan Chang, pp. 198–201 (2009)
16. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing (2003)

17. Bauch, J., Radkowski, R., Zabel, H.: An Explorative Approach to the Virtual Prototyping of Self-optmizing Mechatronic Systems. In: Proceedings of the ProSTEP iViP Science Days - Cross Domain Engineering, Darmstadt (2005)
18. Becker, S., Brenner, C., Brink, C., Dziwok, S., Heinzemann, C., Löffler, R., Pohlmann, U., Schäfer, W., Suck, J., Sudmann, O.: The MechatronicUML Design Method - Process, Syntax, and Semantics. Tech. Rep. tr-ri-12-326, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn (2012)
19. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-agent Systems with JADE. John Wiley & Sons, Hoboken (2007)
20. Ben-Gal, I.: Bayesian Networks. In: Encyclopedia of Statistics in Quality and Reliability (2007)
21. Bengtsson, J.E., Yi, W.: Timed Automata - Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
22. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
23. Berthelot, F., Nouvel, F., Houzet, D.: Partial and Dynamic Reconfiguration of FPGAs: A Top Down Design Methodology for an Automatic Implementation. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium, Rhodes (2006)
24. Bertin, J.: Semiology of Graphics. University of Wisconsin Press, Wisconsin (1983)
25. Beyer, D., Henzinger, T.A., Théoduloz, G., Zufferey, D.: Shape Refinement Through Explicit Heap Analysis. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 263–277. Springer, Heidelberg (2010)
26. Blesken, M., Ruckert, U., Steenken, D., Witting, K., Dellnitz, M.: Multiobjective Optimization for Transistor Sizing of CMOS Logic Standard Cells Using Set-oriented Numerical Techniques. In: Proceedings of the 27th Norchip Conference, Trondheim, pp. 1–4 (2009)
27. Bludau, C., Welp, E.: Semantic Web Services for the Knowledge-based Design of Mechatronic Systems. In: Proceedings of the International Conference on Engineering Design, Melbourne (2005)
28. Böke, C.: Software Synthesis of Real-Time Communication System Code for Distributed Embedded Applications. In: Proceedings of the 6th Annual Australasian Conf. on Parallel and Real-Time Systems, Melbourne (1999)
29. Böke, C.: Automatic Configuration of Real-Time Operating Systems and Real-Time Communication Systems for Distributed Embedded Applications. Ph.D. thesis, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, HNI-Verlagschriftenreihe, Band 142, Paderborn (2003)
30. de Boor, C.: A Practical Guide to Splines. Springer, Heidelberg (2001)
31. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos - A Model-checking Tool for Real-time Systems. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
32. Brutzman, D., Zyda, M., Pullen, M., Morse, K.: XMSF 2002 Findings and Recommendations (2002)
33. Burmester, S., Gehrke, M., Giese, H., Oberthür, S.: Making Mechatronic Agents Resource-Aware to Enable Safe Dynamic Resource Allocation. In: Proceedings of the 4th ACM International Conference on Embedded Software, Pisa (2004)
34. Campos, C., Junge, O., Ober-Blöbaum, S.: Higher Order Variational Time Discretization of Optimal Control Problems. In: Proceedings of the 20th International Symposium on Mathematical Theory of Networks and Systems, Melbourne (2012)

35. Chinapirom, T., Kaulmann, T., Witkowski, U., Rueckert, U.: Visual Object Recognition by 2D-Color Camera and On-Board Information Processing for Minirobots. In: Proceedings of the FIRA Robot World Congress, Busan (2004)
36. Chivukula, R.P., Böke, C., Rammig, F.J.: Customizing the Configuration Process of an Operating System Using Hierarchy and Clustering. In: Proceedings of the 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing, Crystal City, pp. 280–287 (2002)
37. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
38. Commuri, S., Tadigotla, V., Sliger, L.: Task-based Hardware Reconfiguration in Mobile Robots Using FPGAs. Journal of Intelligent and Robotic Systems 49(2), 111–134 (2007)
39. Dahmann, J.S., Fujimoto, R.M., Weatherly, R.M.: The Department of Defense High Level Architecture. In: Proceedings of the 29th Conference on Winter Simulation, Atlanta, pp. 142–149 (1997)
40. David, A., Behrmann, G., Bulychev, P., Byg, J., Chatain, T., Larsen, K.G., Pettersson, P., Rasmussen, J.I., Srba, J., Yi, W., Joergensen, K.Y., Lime, D., Magnin, M., Roux, O.H., Traonouez, L.M.: Tools for Model-Checking Timed Systems. In: Roux, O.H., Jard, C. (eds.) Communicating Embedded Systems - Software and Design, pp. 165–225 (2009)
41. Dell'Aere, A.: Multi-Objective Optimization in Self-Optimizing Systems. In: Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics, Paris, pp. 4755–4760 (2006)
42. Dell'Aere, A.: Numerical Methods for the Solution of Bi-level Multi-objective Optimization Problems. Ph.D. thesis, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, HNI-Verlagschriftenreihe, Paderborn (2008)
43. Dell'Aere, A., Hirsch, M., Klöpper, B., Köster, M., Krupp, A., Krüger, M., Müller, T., Oberthür, S., Pook, S., Priesterjahn, C., Romaus, C., Schmidt, A., Sondermann-Wölke, C., Tichy, M., Vöcking, H., Zimmer, D.: Verlässlichkeit selbstoptimierender Systeme - Potenziale nutzen und Risiken vermeiden, vol. 235. HNI-Verlagsschriftenreihe, Paderborn (2009)
44. Dellnitz, M., Froyland, G., Junge, O.: The Algorithms Behind GAIO - Set Oriented Numerical Methods for Dynamical Systems. In: Fiedler, B. (ed.) Ergodic Theory, Analysis, and Efficient Simulation of Dynamical Systems, pp. 145–174. Springer, Heidelberg (2001)
45. Dellnitz, M., Ober-Blöbaum, S., Post, M., Schütze, O., Thiere, B.: A Multi-objective Approach to the Design of low Thrust Space Trajectories Using Optimal Control. Celestial Mechanics and Dynamical Astronomy 105(1), 33–59 (2009)
46. Dellnitz, M., Schütze, O., Hestermeyer, T.: Covering Pareto Sets by Multilevel Subdivision Techniques. Journal of Optimization Theory and Application 124(1), 113–136 (2005)
47. Dellnitz, M., Witting, K.: Computation of robust Pareto points. International Journal of Computing Science and Mathematics 2(3), 243–266 (2009)
48. DeMillo, R.A., Offutt, A.J.: Constraint-based Automatic Test Data Generation. IEEE Transactions on Software Engineering 17(9) (1991)
49. Ding, L., Davies, D., McMahon, C.A.: The Integration of Lightweight Representation and Annotation for Collaborative Design Representation 19(4), 223–238 (2009)
50. Ding, L., Matthews, J., Mullineux, G.: Annacon - Annotation with Constrains to Support Design. In: Proceedings of the International Conference on Engineering Design, Stanford, pp. 5–48 (2009)

51. Ditze, C.: Towards Operating System Synthesis. Ph.D. thesis, Fachgruppe Entwurf Paralleler Systeme, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 76, Paderborn (2000)
52. Ebied, H.M., Witkowski, U., Rueckert, U., Abdel-Wahab, M.S.: Robot Localization Based on Visual Landmarks. In: Filipe, J., Andrade-Cetto, J., Ferrier, J.L. (eds.) Proceedings of the 5th IEEE International Conference on Informatics in Control, Automation and Robotics, Funchal, pp. 49–53 (2008)
53. Eckardt, T., Heinzemann, C., Henkler, S., Hirsch, M., Priesterjahn, C., Schäfer, W.: Modeling and Verifying Dynamic Communication Structures Based on Graph Transformations. Computer Science - Research and Development 28, 3–22 (2013)
54. Eckardt, T., Henkler, S.: Component Behavior Synthesis for Critical Systems. In: Giese, H. (ed.) ISARCS 2010. LNCS, vol. 6150, pp. 52–71. Springer, Heidelberg (2010)
55. Eckardt, T., Henkler, S.: Synthesis of Reconfiguration Charts. Tech. Rep. tr-ri-10-314, Software Engineering Group, University of Paderborn (2010)
56. Esau, N., Krüger, M., Rasche, C., Beringer, S., Kleinjohann, L., Kleinjohann, B.: Hierarchical Hybrid Planning for a Self-Optimizing Active Suspension System. In: Proceedings of the 7th IEEE Conference in Industrial Electronics and Applications, Singapore (2012)
57. Estler, H.C., Wehrheim, H.: Heuristic Search-based Planning for Graph Transformation Systems. In: Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling, Freiburg, pp. 54–61 (2011)
58. Feng, H.H., Kolesnikov, O.M., Fogla, P., Lee, W., Gong, W.: Anomaly Detection Using Call Stack Information. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy, Berkeley (2003)
59. FG Rammig, University of Paderborn: ORCOS - Organic Reconfigurable Operating System, `https://orcos.cs.uni-paderborn.de/doxygen/html` (accessed August 12, 2013)
60. Flaßkamp, K., Murphey, T., Ober-Blöbaum, S.: Switching Time Optimization in Discretized Hybrid Dynamical Systems. In: Proceedings of the 51th IEEE Conference on Decision and Control, Maui, pp. 707–712 (2012)
61. Flaßkamp, K., Ober-Blöbaum, S.: Variational Formulation and Optimal Control of Hybrid Lagrangian systems. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, Chicago, pp. 241–250. ACM Press, New York (2011)
62. Flaßkamp, K., Ober-Blöbaum, S., Kobilarov, M.: Solving Optimal Control Problems by Exploiting Inherent Dynamical Systems Structures. Journal of Nonlinear Science 22(4), 599–629 (2012)
63. Flaßkamp, K., Ober-Blöbaum, S., Ringkamp, M., Schneider, T., Schulte, C., Böcker, J.: Berechnung optimaler Stromprofile für einen 6-phasigen, geschalteten Reluktanzantrieb. In: Tagungsband Vom 8. Paderborner Workshop Entwurf mechatronischer Systeme. Heinz Nixdorf Institut Verlagsschriftreihe, Paderborn (2011)
64. Flaßkamp, K., Timmermann, J., Ober-Blöbaum, S., Dellnitz, M., Trächtler, A.: Optimal Control on Stable Manifolds for a Double Pendulum. In: Applied Mathematics and Mechanics, vol. 12, pp. 723–724. Springer, Heidelberg (2012)
65. Fox, M., Long, D.: PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Jornal of Artificial Intelligence Research, 189–208 (2003)
66. Frazzoli, E., Dahleh, M.A., Feron, E.: Maneuver-based Motion Planning for Nonlinear Systems with Symmetries. IEEE Trans. on Robotics 21(6), 1077–1091 (2005)

67. Galea, A., Borg, J., Grech, A., Farrugia, P.: Towards Intelligent Design Tools for Microscale components. In: Proceedings of the International Conference on Engineering Design, Stanford, pp. 5–84 (2009)
68. Gausemeier, J., Frank, U., Donoth, J., Kahl, S.: Specification Technique for the Description of Self-optimizing Mechatronic Systems. Research in Engineering Design 20(4), 201–223 (2009)
69. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
70. Gausemeier, J., Schäfer, W., Greenyer, J., Kahl, S., Pook, S., Rieke, J.: Management of Cross-Domain Model Consistency During the Development of Advanced Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
71. Geiger, C., Lehrenfeld, G., Müller, W.: Authoring Communicating Agents in Virtual Environments. In: Proceedings of the Computer Human Interaction, Adelaide, pp. 22–29 (1998)
72. Geisler, J., Witting, K., Trächtler, A., Dellnitz, M.: Multiobjective Optimization of Control Trajectories for the Guidance of a Rail-bound Vehicle. In: Proceedings of the 17th IFAC World Congress, Seoul (2008)
73. Geisler, J., Witting, K., Trächtler, A., Dellnitz, M.: Multiobjective Optimization of Control Trajectories for the Guidance of a Rail-bound Vehicle. In: Proceedings of the 17th World Congress International Federation of Automatic Control, Milano (2008)
74. Ghallab, M., Nau, D., Traverso, P.: Automated Planning - Theory and Practice. Elsevier, Amsterdam (2004)
75. Giese, H., Tichy, M., Burmester, S., Schäfer, W., Flake, S.: Towards the Compositional Verification of Real-time UML Designs. In: Proceedings of the 9th European Software Engineering Conference Held Jointly with the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Helsinki, pp. 38–47. ACM Press, New York (2003)
76. Gill, P.E., Jay, L.O., Leonard, M.W., Petzold, L.R., Sharma, V.: An SQP Method for the Optimal Control of Large-scale Dynamical Systems. Journal of Computational and Applied Mathematics 120, 197–213 (2000)
77. Gilles, K., Groesbrink, S., Baldin, D., Kerstan, T.: Proteus Hypervisor - Full Virtualization and Paravirtualization for Multi-Core Embedded Systems. In: Proceedings of the International Embedded Systems Symposium, Paderborn (2013)
78. Greenyer, J., Kindler, E.: Comparing Relational Model Transformation Technologies: Implementing Query/View/Transformation with Triple Graph Grammars. Software and Systems Modeling 9, 21–46 (2010)
79. Greenyer, J., Pook, S., Rieke, J.: Preventing Information Loss in Incremental Model Synchronization by Reusing Elements. In: Proceedings of the 7th European Conference on Modelling Foundations and Applications, Birmingham (2011)
80. Griese, B., Oberthür, S., Porrmann, M.: Component Case Study of a Self-optimizing RCOS/RTOS System: A Reconfigurable Network Service. In: Proceedings of the International Embedded Systems Symposium - From Specification to Embedded Systems Application, Manaos, pp. 267–277 (2005)
81. Griewank, A., Walther, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Society for Industrial and Applied Mathematics, Philadelphia (2008)

82. Groesbrink, S.: A First Step Towards Real-time Virtual Machine Migration in Heterogeneous Multi-Processor Systems. In: Proceedings of the 1st Joint Symposium on System-Integrated Intelligence, Hannover (2012)
83. Groesbrink, S.: Basics of Virtual Machine Migration on Heterogeneous Architectures for Self-optimizing Mechatronic Systems - Necessary Conditions and Implementation Issues. In: Production Engineering Research & Development (11740) (2012)
84. Guckenheimer, J., Holmes, P.: Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields. In: Applied Mathematical Sciences, vol. 42, Springer, Heidelberg (1983)
85. Guleyupoglu, S., Ng, H.: Distributed Collaborative Virtual Reality Framework for System Prototyping and Training. In: Proceedings of the RTO IST Symposium on New Information Processing Techniques for Military Systems, Istanbul (2000)
86. Gutiérrez, M., Vexo, F., Thalmann, D.: Stepping into Virtual Reality. Springer, Heidelberg (2008)
87. Hagemeyer, J., Kettelhoit, B., Koester, M., Porrmann, M.: Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs. In: International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas (2007)
88. Hagemeyer, J., Kettelhoit, B., Köster, M., Porrmann, M.: A Design Methodology for Communication Infrastructures on Partially Reconfigurable FPGAs. In: Proceedings of the 17th International Conference on Field Programmable Logic and Applications, Amsterdam (2007)
89. Hampton, M., Petithomme, S.: Leveraging a Commercial Mutation Analysis Tool for Research. In: Proceedings of the Testing Academic & Industrial Conference Practice and Research Techniques, Windsor (2007)
90. Heckel, R., Thöne, S.: Behavioral Refinement of Graph Transformation-based Models. In: Proceedings of the Workshop on Software Evolution through Transformations: Model-based vs. Implementation-level Solutions, Rom, pp. 101–111 (2005)
91. Heinzemann, C., Henkler, S.: Reusing Dynamic Communication Protocols in Self-Adaptive Embedded Component Architectures. In: Proceedings of the 14th International Symposium on Component Based Software Engineering, Boulder, pp. 109–118 (2011)
92. Heinzemann, C., Henkler, S.: Timed Story Driven Modeling. Tech. Rep. tr-ri-11-326, University of Paderborn (2011)
93. Heinzemann, C., Pohlmann, U., Rieke, J., Schäfer, W., Sudmann, O., Tichy, M.: Generating Simulink and Stateflow Models From Software Specifications. In: Proceedings of the 12h International Design Conference DESIGN, Dubrovnik (2012)
94. Heinzemann, C., Priesterjahn, C., Becker, S.: Towards Modeling Reconfiguration in Hierarchical Component Architectures. In: Proceedings of the 15th ACM SigSoft International Symposium on Component-Based Software Engineering, Bertinoro, pp. 23–28 (2012)
95. Heinzemann, C., Rieke, J., Schäfer, W.: Simulating self-adaptive component-based systems using matlab/simulink. In: Proceedings of the 7th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2013. IEEE Computer Society Press (2013)
96. Henkler, S., Meyer, J., Schäfer, W., Nickel, U.: Reverse Engineering mechatronischer Systeme. In: Proceedings of the 7th Paderborner Workshop Entwurf Mechatronischer Systeme, Paderborn (2010)

97. Henkler, S., Meyer, J., Schäfer, W., Nickel, U.A., von Detten, M.: Legacy Component Integration by the Fujaba Real-time Tool Suite. In: Proceedings of the 32nd International Conference on Software Engineering, Cape Town, vol. 2, pp. 267–270 (2010)
98. Henzinger, T.A.: The Theory of Hybrid Automata. In: Logic in Computer Science, p. 278 (1996)
99. Herbrechtsmeier, S., Witkowski, U., Rückert, U.: BeBot - A Modular Mobile Miniature Robot Platform Supporting Hardware Reconfiguration and Multi-standard Communication. In: Kim, J.-H., Ge, S.S., Vadakkepat, P., Jesse, N., Al Manum, A., Puthusserypady, K.S., Rückert, U., Sitte, J., Witkowski, U., Nakatsu, R., Braunl, T., Baltes, J., Anderson, J., Wong, C.-C., Verner, I., Ahlgren, D. (eds.) Progress in Robotics. CCIS, vol. 44, pp. 346–356. Springer, Heidelberg (2009)
100. Hillermeier, C.: Nonlinear Multiobjective Optimization - A Generalized Homotopy Approach. Birkhäuser (2001)
101. Hitzler, P., Krötzsch, M., Rudolph, S., Sure, Y.: Semantic Web - Grundlagen. Springer, Heidelberg (2008)
102. Hölscher, C., Keßler, J.H., Krüger, M., Trächtler, A., Zimmer, D.: Hierarchical Optimization of Coupled Self-optimizing Systems. In: Proceedings of the 10th IEEE International Conference on Industrial Informatics, Beijing (2012)
103. Horta, E.L., Lockwood, J.W.: PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs). Tech. rep. (2001)
104. Howden, W.E.: Weak Mutation Testing and Completeness of Test Sets. IEEE Transactions on Software Engineering 8(4) (1982)
105. Hussmann, M., Thies, M., Kastens, U., Purnaprajna, M., Porrmann, M., Rueckert, U.: Compiler-driven Reconfiguration of Multiprocessors. In: Proceedings of the Workshop on Application Specific Processors, Salzburg, pp. 3–10 (2007)
106. for Intelligent Physical Agents, F.: FIPA Propose Interaction Protocol Specification (2002), `http://www.fipa.org/specs/fipa00036/SC00036H.pdf` (accessed May 8, 2012)
107. Jantsch, A., Tenhunen, H.: Networks on Chip. Kluwer Academic Publishers, Dordrecht (2003)
108. Jennings, N.R., Wooldrige, M.: Applying Agent Technology. Applied Artificial Intelligence 9(4), 357–369 (1995)
109. Jungeblut, T., Ax, J., Porrmann, M., Rueckert, U.: A TCMS-based Architecture for GALS NoCs. In: Proceedings of the IEEE International Symposium on Circuits and Systems, Seoul (2012)
110. Jungeblut, T., Liss, C., Porrmann, M., Rueckert, U.: Design-space Exploration for Flexible WLAN Hardware. In: Zorba, N., Skianis, C., Verikoukis, C. (eds.) Cross Layer Designs in WLAN Systems, pp. 521–564. Troubador Publishing, Leicester (2011)
111. Jungmann, A., Kleinjohann, B., Kleinjohann, L., Bieshaar, M.: Efficient Color-Based Image Segmentation and Feature Classification for Image Processing in Embedded Systems. In: Proceedings of the 4th International Conference on Resource Intensive Applications and Services, St. Maarten (2012)
112. Kalte, H., Lee, G., Porrmann, M., Rückert, U.: REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems. In: Proceedings of the 19th International Parallel and Distributed Processing Symposium - Reconfigurable Architectures Workshop (2005)
113. Kastenberg, H., Rensink, A.: Model Checking Dynamic States in GROOVE. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 299–305. Springer, Heidelberg (2006)

114. Katzenbach, A., Haasis, S.: Virtual and Mixed Reality in a SOA Based Engineering Environment. In: Proceedings of the CIRP Design Conference Design Synthesis, Enschede (2008)
115. Kerstan, T., Oertel, M.: Design of a Real-time Optimized Emulation Method. In: Proceedings of the Design, Automation and Test in Europe, Dresden (2010)
116. Kettelhoit, B., Porrmann, M.: A Layer Model for Systematically Designing Dynamically Reconfigurable Systems. In: Proceedings of the 16th International Conference on Field Programmable Logic and Applications, Madrid (2006)
117. Klinker, G., Dutoit, A., Bauer, M., Bayer, J., Novak, V.: Fata Morgana - A Presentation System for Product Design. In: Proceedings of the International Symposium on Mixed and Augmented Reality, Darmstadt (2002)
118. Klöpper, B.: Ein Beitrag zur Verhaltensplanung für interagierende intelligente mechatronische Systeme in nicht-deterministischen Umgebungen. Ph.D. thesis, Fakultät für Wirtschaftswissenschaften, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 253, Paderborn (2009)
119. Klöpper, B., Aufenanger, M., Adelt, P.: Planning for Mechatronics Systems - Architechture, Methods and Case Study. Engineering Applications of Artificial Intelligence 25(1), 174–188 (2012)
120. Koester, M., Kalte, H., Porrmann, M.: Run-time Defragmentation for Partially Reconfigurable Systems. In: Proceedings of the IFIP International Conference on Very Large Scale Integration, Madrid, pp. 109–115 (2005)
121. Koester, M., Kalte, H., Porrmann, M.: Task Placement for Heterogeneous Reconfigurable Architectures. In: Proceedings of the IEEE 2005 Conference on Field-Programmable Technology, Singapore, pp. 43–50 (2005)
122. Koester, M., Kalte, H., Porrmann, M.: Task Placement for Heterogeneous Reconfigurable Architectures. In: Proceedings of the IEEE 2005 Conference on Field-Programmable Technology, Singapore, pp. 43–50 (2005)
123. Koester, M., Luk, W., Hagemeyer, J., Porrmann, M.: Design Optimizations to Improve Placeability of Partial Reconfiguration Modules. In: Proceedings of the International Conference on Design, Automation and Test in Europe, Nice (2009)
124. Koester, M., Porrmann, M., Rückert, U.: Placement-oriented Modeling of Partially Reconfigurable Architectures. In: Proceedings of the 19th International Parallel and Distributed Processing Symposium - Reconfigurable Architectures Workshop, Phoenix (2005)
125. Köpper, B., Sondermann-Wölke, C., Romaus, C.: Probabilistic Planning for Predictive Condition Monitoring and Adaptation within the Self-Optimizing Energy Management of an Autonomous Railway Vehicle. Journal for Robotics and Mechatronics 24, 5–15 (2012)
126. Korf, S., Cozzi, D., Koester, M., Hagemeyer, J., Porrmann, M., Rückert, U., Santambrogio, M.D.: Automatic HDL-based Generation of Homogeneous Hard Macros for FPGAs. In: Proceedings of the IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, Salt Lake City, pp. 125–132 (2011)
127. Kramer, J., Magee, J.: Analysing Dynamic Change in Software Architectures: A Case Study. In: Proceedings of the International Conference on Configurable Distributed Systems, Annapolis (1998)
128. Krause, F.L., Jansen, H., Kind, C., Rothenburg, U.: Virtual Product Development as an Engine for Innovation. In: Krause, F.L. (ed.) The Future of Product Development, pp. 703–713. Springer, Heidelberg (2007)

129. Krüger, M., Trächtler, A.: Approximation of Pareto-optimal Systems Using Parametric Model-order Reduction. In: 7th Vienna International Conference on Mathematical Modelling, Wien
130. Krüger, M., Witting, K., Dellnitz, M., Trächtler, A.: Robust Pareto Points with Respect to Crosswind of an Active Suspension System. In: Proceedings of the 1st Joint International Symposium on System-Integrated Intelligence, Hannover (2012)
131. Krüger, M., Witting, K., Trächtler, A., Dellnitz, M.: Parametric Model-order Reduction in Hierarchical Multiobjective Optimization of Mechatronic Systems. In: Proceedings of the 18th IFAC World Congress, Milano (2011)
132. Leyendecker, S., Ober-Blöbaum, S.: A Variational Approach to Multirate Integration. In: Proceedings of the 4th European Conference on Computational Mechanics, Paris (2010)
133. Leyendecker, S., Ober-Blöbaum, S.: A Variational Approach to Multirate Integration for Constrained Systems. In: Fisette, P., Samin, J.C. (eds.) Proceedings of the ECCOMAS Thematic Conference: Multibody Dynamics: Computational Methods and Applications, Brüssel (2011)
134. Leyendecker, S., Ober-Blöbaum, S., Marsden, J.E., Ortiz, M.: Discrete Mechanics and Optimal Control for Constrained Systems. Optimal Control, Applications and Methods 31(6), 505–528 (2010)
135. Li, C., McMahon, C., Newnes, L.: Annotation in Design Processes: Classification of Approcches. In: Proceedings of the International Conference on Engineering Design, Stanford, pp. 8–262 (2009)
136. Li, L., Littman, M.L., Littman, L.: Prioritized Sweeping Converges to the Optimal Value Function. Tech. Rep. DCS-TR-631 (2008)
137. Loginov, A., Reps, T., Sagiv, M.: Abstraction Refinement via Inductive Learning. In: Proceedings of the 17th International Conference on Computer Aided Verification, Edinburgh, pp. 519–533 (2005)
138. Luetkemeier, S., Porrmann, M., Jungeblut, T., Rueckert, U.: A 200 mV 32-bit Subthreshold Processor with Adaptive Supply Voltage Control. In: Proceedings of the 2012 IEEE International Solid-state Circuits Conference, San Francisco, pp. 484–485 (2012)
139. Lynch, N.A.: Distributed Algorithms, 1st edn. Morgan Kaufmann, Burlington (1997)
140. Marsden, J.E., Ratiu, T.S.: Introduction to Mechanics and Symmetry, 2nd edn. Springer, Heidelberg (1999)
141. Marsden, J.E., West, M.: Discrete Mechanics and Variational Integrators. Acta Numerica 10, 357–514 (2001)
142. Mendez, G., de Antonio, A.: An Agent-based Architecture for Collaborative Virtual Environments for Training. In: Proceedings of the 5th WSEAS Int. Conf. on Multimedia, Internet and Video Technologies, Corfu, pp. 29–34 (2005)
143. Mescheder, D., Tuyls, K., Kaisers, M.: POMDP Opponent Models for Best Response Behavior. In: Proceedings of the 23rd Benelux Conference on Artificial Intelligence, Gent (2011)
144. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1982)
145. Moctezuma Eugenio, J.C., Arias Estrada, M.: Hardware/Software FPGA Architecture for Robotics Applications. In: Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications, Karlsruhe, pp. 27–38 (2009)
146. Moore, A., Ober-Blöbaum, S., Marsden, J.E.: Trajectory Design Combining Invariant Manifolds with Discrete Mechanics and Optimal Control. Journal of Guidance, Control, and Dynamics 35(5), 1507–1525 (2012)

147. Moore, A.W., Atkeson, C.G.: Prioritized Sweeping - Reinforcement Learning with less Data and less Time. Machine Learning 13(1), 103–130 (1993)
148. Münch, E., Gambuzza, A., Paiz, C., Pohl, C., Porrmann, M.: FPGA-in-the-Loop Simulations with CAMEL-View. In: Proceedings of the 7th International Heinz Nixdorf Symposium, Paderborn (2008)
149. Nava, F., Sciuto, D., Santambrogio, M.D., Herbrechtsmeier, S., Porrmann, M., Witkowski, U., Rueckert, U.: Applying Dynamic Reconfiguration in the Mobile Robotics Domain - A Case Study on Computer Vision Algorithms. ACM Transactions on Reconfigurable Technology and Systems 4(3), 29:1–29:22 (2011)
150. Niemann, J.C., Puttmann, C., Porrmann, M., Rückert, U.: Resource Efficiency of the GigaNetIC Chip Multiprocessor Architecture. Journal of Systems Architecture (JSA), Special Issue on Architectural Premises for Pervasive Computing 53(5-6), 285–299 (2007)
151. Ober-Blöbaum, S., Junge, O., Marsden, J.E.: Discrete Mechanics and Optimal Control: An Analysis. Control, Optimisation and Calculus of Variations 17(2), 322–352 (2011)
152. Ober-Blöbaum, S., Ringkamp, M., Zum Felde, G.: Solving Multiobjective Optimal Control Problems in Space Mission Design using Discrete Mechanics and Reference Point Techniques. In: Proceedings of the 51th IEEE Conference on Decision and Control, Maui, pp. 5711–5716 (2012)
153. Ober-Blöbaum, S., Walther, A.: Computation of Derivatives for Structure Preserving Optimal Control Using Automatic Differentiation. Proceedings of Applied Mathematics and Mechanics 10(1), 585–586 (2010)
154. Oberthür, S.: Towards an RTOS for Self-optimizing Mechatronic Systems. Ph.D. thesis, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, HNI-Verlagschriftenreihe, Paderborn (2009)
155. Oberthür, S.: Towards an RTOS for Self-optimizing Mechatronic Systems. Ph.D. thesis, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, HNI-Verlagschriftenreihe, Paderborn (2010)
156. Oberthür, S., Böke, C.: Flexible Resource Management - A Framework for Self-optimizing Real-time Systems. In: Gao, G.R., Kopetz, H., Kleinjohann, L., Rettberg, A. (eds.) Proceedings of IFIP Working Conference on Distributed and Parallel Embedded Systems, Toulouse (2004)
157. Oberthür, S., Zaramba, L., Lichte, H.S.: Flexible Resource Management for Self-X Systems: An Evaluation. In: Proceedings of the 1st IEEE Workshop on Self-Organizing Real-Time Systems, Carmona (2010)
158. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Engineering Design - A Systematic Approach, 3rd edn. Springer, Heidelberg (2007)
159. Paiz, C., Hagemeyer, J., Pohl, C., Porrmann, M., Rückert, U., Schulz, B., Peters, W., Böcker, J.: FPGA-Based Realization of Self-Optimizing Drive-Controllers. In: Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, Porto (2009)
160. Panzer, H., Mohring, J., Eid, R., Lohmann, B.: Parametric Model Order Reduction by Matrix Interpolation. at - Automatisierungstechnik 58, 475–484 (2010)
161. Payne, T.: Agent-based Team Aiding in a Time Critical Task. In: Proceeding of the 44rd Hawaii International Conference on System Sciences, Maui, vol. 1 (2000)
162. Pohl, C., Paiz, C., Porrmann, M.: A Hardware-in-the-Loop Design Environment for FPGAs. In: Proceedings of the Design, Automation and Test in Europe, München (2008)

163. Pohl, C., Paiz, C., Porrmann, M.: vMAGIC - VHDL Manipulation and Automation for Reliable System Development. In: Proceedings of the 3rd International Workshop on Reconfigurable Computing Education, Karlsruhe (2008)
164. Porrmann, M., Hagemeyer, J., Pohl, C., Romoth, J., Strugholtz, M.: RAPTOR - A Scalable Platform for Rapid Prototyping and FPGA-based Cluster Computing. In: Proceedings of the Parallel Computing: From Multicores and GPUs to Petascale, Lyon, pp. 592–599 (2010)
165. Porrmann, M., Purnaprajna, M., Puttmann, C.: Self-optimization of MPSoCs Targeting Resource Efficiency and Fault Tolerance. In: Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, pp. 467–473 (2009)
166. Priesterjahn, C.: Hazard Analysis of Self-optimizing Mechatronic Systems. In: Proceedings of the Doctoral Symposium of the 7th Joint Meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Amsterdam (2009)
167. Purnaprajna, M., Porrmann, M., Rueckert, U.: Run-time Reconfigurability in Embedded Multiprocessors. SIGARCH Computer Architecture News 37(2), 30–37 (2009)
168. Purnaprajna, M., Porrmann, M., Rueckert, U., Hussmann, M., Thies, M., Kastens, U.: Runtime Reconfiguration of Multiprocessors Based on Compile-time Analysis. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 3(3), 17:1–17:25 (2010)
169. Purnaprajna, M., Puttmann, C., Porrmann, M.: Power Aware Reconfigurable Multiprocessor for Elliptic Curve Cryptography. In: Proceedings of the Conference on Design, Automation and Test in Europe, München, pp. 1462–1467 (2008)
170. Puttmann, C., Niemann, J.C., Porrmann, M., Rückert, U.: GigaNoC – A Hierarchical Network-on-Chip for Scalable Chip-Multiprocessors. In: Proceedings of the 10th EUROMICRO Conference on Digital System Design, Lübeck, pp. 495–502 (2007)
171. Radkowski, R.: Towards Semantic Virtual Prototypes for Automatic Model Combination. In: Proceedings of the 20th CIRP Design Conference, Global Product Development, Nantes (2010)
172. Radkowski, R., Waßmann, H.: Augmented Reality-based Approach for the Visual Analysis of Intelligent Mechatronic Systems. In: Proceedings of the Workshop at the IDETC/CIE Design Engineering Technical Conference & Computer and Information in Engineering Conference, New York (2008)
173. Radkowski, R., Waßmann, H.: Software-agent Supported Virtual Experimental Environment for Virtual Prototypes of Mechatronic Systems. In: Proceedings of the ASME 2010 World Conference on Innovative Virtual Reality, Ames (2010)
174. Rana, V., Santambrogio, M., Sciuto, D., Kettelhoit, B., Koester, M., Porrmann, M., Rückert, U.: Partial Dynamic Reconfiguration in a Multi-FPGA Clustered Architecture Based on Linux. In: Proceedings of the 21st International Parallel and Distributed Processing Symposium: Reconfigurable Architectures Workshop, Long Beach (2007)
175. Reinold, P., Nachtigal, V., Trächtler, A.: An Advanced Electric Vehicle for the Development and Test of New Vehicle-Dynamics Control Strategies. In: Proceedings of the 6th IFAC Symposium on Advances in Automotive Control AAC, München (2010)
176. Reps, T., Sagiv, M., Loginov, A.: Finite Differencing of Logical Formulas for Static Analysis (ESOP). In: Proceedings of European Symposium on Programming, Las Vegas, vol. 32, pp. 393–412 (2003)
177. Restrepo, J.: A Visual Lexicon to Handle Semantic Similarity in Design Precedents. In: Proceedings of the 16th International Conference on Engineering Design, Paris (2007)

178. Reynolds, C.W.: Steering Behaviors For Autonomous Characters. In: Proceedings of Game Developers Conference, San Jose, pp. 763–782 (1999)
179. Richter, U., Mnif, M., Branke, J., Müller-Schloer, C., Schmeck, H.: Towards a Generic Observer/Controller Architecture for Organic Computing. In: Tagungsband vom 36, pp. 112–119. Jahrestagung der Gesellschaft für Informatik - Informatik für Menschen, Dresden (2006)
180. Rieke, J., Dorociak, R., Sudmann, O., Gausemeier, J., Schäfer, W.: Management of Cross-domain Model Consistency for Behavioral Models of Mechatronic Systems. In: Proceedings of the 12th International Design Conference, Dubrovnik (2012)
181. Ringkamp, M., Ober-Blöbaum, S., Dellnitz, M., Schütze, O.: Handling High Dimensional Problems with Multi-objective Continuation Methods via Successive Approximation of the Tangent Space. Engineering Optimization 44(9), 1117–1146 (2012)
182. Ringkamp, M., Walther, A., Reinold, P., Witting, K., Dellnitz, M., Trächtler, A.: Using Algorithmic Differentiation for the Multiobjective Optimization of a Test Vehicle. In: Proceedings of EVOLVE, Mexico City (2012)
183. Romaus, C., Bocker, J., Witting, K., Seifried, A., Znamenshchykov, O.: Optimal Energy Management for a Hybrid Energy Storage System Combining Batteries and Double Layer Capacitors. In: Proceedings of the Energy Conversion Congress and Exposition, San Jose, pp. 1640–1647 (2009)
184. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation. Foundations, vol. 1. World Scientific Publishing Co. Inc., River Edge (1997)
185. Sagiv, M., Reps, T., Wilhelm, R.: Parametric Shape Analysis via 3-valued Logic. ACM Transactions on Programming Languages and Systems 24(3), 217–298 (2002)
186. Schenk, M., Straßburger, S., Kissner, H.: Combining Virtual Reality and Assembly Simulation for Production Planning and Worker Qualification. In: Zaeh, M., Reinhart, G. (eds.) Proceedings of the International Conference on Changeable, Agile, Reconfigurable and Virtual Production, München, pp. 411–414 (2005)
187. Schneider, T., Schulz, B., Henke, C., Witting, K., Steenken, D., Böcker, J.: Energy Transfer via Linear Doubly-fed Motor in Different Operating Modes. In: Proceedings of the International Electric Machines and Drives Conference, Miami, pp. 598–605 (2009)
188. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
189. Schütze, O., Dell'Aere, A., Dellnitz, M.: On Continuation Methods for the Numerical Treatment of Multi-objective Optimization Problems. In: Proceedings of the Practical Approaches to Multi-objective Optimization, Dagstuhl (2005)
190. Schütze, O., Witting, K., Ober-Blöbaum, S., Dellnitz, M.: Set Oriented Methods for the Numerical Treatment of Multi-objective Optimization Problems. In: Tantar, E., Tantar, A.-A., Bouvry, P., Del Moral, P., Legrand, P., Coello Coello, C.A., Schütze, O. (eds.) EVOLVE- A bridge between Probability. SCI, vol. 447, pp. 187–219. Springer, Heidelberg (2013)
191. Serrestou, Y., Beroulle, V., Robach, C.: Functional Verification of RTL Designs Driven by Mutation Testing Metrics. In: Proceedings of the 10th Euromicro Conference on Digital System Design, Lebeck, pp. 222–227 (2007)
192. Spors, K., Martin, A., Leetz, A.: Möglichkeiten fotorealistischer Visualisierungen im Produktionsprozess eines Automobils. Automobiltechnische Zeitschrift 3, 1–8 (2009)

193. Steenken, D., Wehrheim, H., Wonisch, D.: Sound and Complete Abstract Graph Transformation. In: Proceedings of the Brazilian Symposium on Formal Methods, Sao Paulo, pp. 92–107 (2011)
194. Steenken, D., Wonisch, D.: Using Shape Analysis to verify Graph Transformations in Model Driven Design. In: Proceedings of the 9th IEEE International Conference on Industrial Informatics, Lisbon, pp. 457–462 (2011)
195. Groesbrink, S., Baldin, D.: Towards Adaptive Resource Management for Virtualized Real-Time Systems. In: Proceedings of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems, Beijing (2012)
196. Suck, J., Heinzemann, C., Schäfer, W.: Formalizing Model Checking on Timed Graph Transformation Systems. Tech. Rep. tr-ri-11-316, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn (2011)
197. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (1998)
198. Szyperski, C.: Component Software: Beyond Object-oriented Programming. Addison-Wesley, Bonn (1998)
199. Thiere, B., Ober-Blöbaum, S., Pergola, P.: Detecting Initial Guesses for Trajectories in the (P)CRTBP. In: Proceedings of the AIAA/AAS Astrodynamics Specialist Conference, Toronto (2010)
200. Tichy, M., Henkler, S., Holtmann, J., Oberthür, S.: Component Story Diagrams: A Transformation Language for Component Structures in Mechatronic Systems. In: Proceedings of the 4th Workshop on Object-oriented Modeling of Embedded Real-time Systems, Paderborn, pp. 27–39 (2008)
201. Timmermann, R., Horenkamp, C., Dellnitz, M., Keßler, J.H., Trächtler, A.: Optimale Umschaltstrategien bei Aktorausfall mit Pfadverfolgungstechniken. In: Gausemeier, J., Rammig, F.J., Schäfer, W., Trächtler, A. (eds.) Tagungsband vom 9. Paderborner Workshop Entwurf mechatronischer Systeme. HNI-Verlagsschriftenreihe, Paderborn (2013)
202. Tripakis, S., Yovine, S.: Analysis of Timed Systems Using Time-abstracting Bisimulations. Formal Methods in System Design 18(1), 25–68 (2001)
203. University of Paderborn: TGG Interpreter Tool Suite (2012), `http://www.cs.uni-paderborn.de/index.php?id=tgg-interpreter` (accessed August 13, 2013)
204. Wasson, C.S.: System Analysis, Design, and Development. John Wiley & Sons, Hoboken (2006)
205. Watkins, C.J.C.H., Dayan, P.: Q-Learning. Machine Learning 8(3-4), 279–292 (1992)
206. Witting, K.: Numerical Algorithms for the Treatment of Parametric Multiobjective Optimization Problems and Applications. Ph.D. thesis, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, HNI-Verlagschriftenreihe, Paderborn (2011)
207. Witting, K., Ober-Blöbaum, S., Dellnitz, M.: A Variational Approach to Define Robustness for Parametric Multiobjective Optimization Problems. Journal of Global Optimization (2012)
208. Witting, K., Schulz, B., Dellnitz, M., Böcker, J., Fröhleke, N.: A new Approach for Online Multiobjective Optimization of Mechatronic Systems. International Journal on Software Tools for Technology Transfer STTT 10(3), 223–231 (2008)
209. Wittke, M.: AR in der PKW-Entwicklung bei Volkswagen. In: Schenk, M. (ed.) Tagungsband zur 4. Fachtagung zu Virtual RealityIFF-Wissenschaftstage - Virtual Reality und Augmented Reality zum Planen, Testen und Betreiben technischer Systeme, Magdeburg (2007)

210. Wolf, W., Jerraya, A., Martin, G.: Multiprocessor System-on-Chip (MPSoC) Technology. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(10), 1701–1713 (2008)
211. Wonisch, D.: Increasing the Preciseness of Shape Analysis for Graph Transformation Systems. Ph.D. thesis, Institut für Informatik, Universität of Paderborn (2010)
212. Ye, J., Badiyani, S., Raja, V., Schlegel, T.: Applications of Virtual Reality in Product Design Evaluation. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part IV, HCII 2007. LNCS, vol. 4553, pp. 1190–1199. Springer, Heidelberg (2007)
213. Zhang, J., Cheng, B.H.C.: Model-based Development of Dynamically Adaptive Software. In: Proceedings of the 28th International Conference on Software Engineering, Shanghai (2006)
214. Zilberstein, S.: Using Anytime Algorithms in Intelligent Systems. AI Magazine 17(3), 73–83 (1996)

# Chapter 6
# Summary and Outlook

Jürgen Gausemeier and Mareen Vaßholz

The increasing development of information and communication technology enables mechatronic systems with inherent partial intelligence, so called self-optimizing systems. Their behavior is formed by the communication and cooperation of intelligent system elements. Self-optimization describes the endogenous adaptation of the system's objectives due to changing operation conditions and the resulting autonomous adjustment of the system's behavior. Self-optimization therefore opens up fascinating prospects for the development of future mechatronic systems, which meet the increasing requirements on such systems. At the same time the development of self-optimizing systems sets new requirements on the design methodology, due to the involvement of different domains such as mechanical, electrical/electronic, control, software engineering and experts from higher mathematics and artificial intelligence. This leads to an increasing design complexity and requires an effective cooperation and communication between the developers. The approach of the Collaborative Research Center (CRC) 614 for the development of self-optimizing systems presented in this book, overcomes the shortcomings of the existing design methodologies. It provides a design methodology consisting of a reference process, tools and methods. It makes the self-optimization specific expertise available for the developers and enables them to develop these systems independently.

After introducing the paradigm of self-optimization and deriving the challenges for their development in Chap. 1 of this book, the various application example underline the capabilities of self-optimizing systems. Based on these application examples, presented in Chap. 2, different methods, developed within the CRC 614, were evaluated. The implementation of self-optimization led for example to minimized energy losses or an increased comfort for the passenger in the rail vehicle RailCab. All application examples clarify the high potential benefit of self-optimizing systems. To take advantage of these benefits a design methodology for the development of self-optimizing systems is presented in Chap. 3. The reference process is divided into two phases, the domain-spanning conceptual design and the domain-specific design and development. Within the domain-spanning conceptual design the principle solution is developed which generates a common understanding of the

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*,
Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_6, © Springer-Verlag Berlin Heidelberg 2014

system for all domains involved. The methods and tools that are necessary to develop the principle solution are presented in Chap. 4. Based on the principle solution the domain-specific design and development can be initiated. During this phase the domains involved work in parallel with their domain specific tools and methods. For this phase the reference process points out the self-optimization specific tasks and is therefore not intended as a substitute but as a supplement to the existing domain-specific development processes. It compromises the domains mechanical, control, software and electrical/electronic engineering as for classical mechatronic systems, but also experts for the (sub)system optimization where the self-optimization process is implemented. For this task expertise from higher mathematics and artificial intelligence are needed. For a specific development task the design and development phase needs to be tailored individually due to development objectives and organizational conditions. For this we applied the paradigm of self-optimization to the management of development processes (cf. Sect. 3.4). The self-optimization specific methods, tools and expertise that are needed for the design and development of self-optimizing systems are presented in Chap. 5. To ensure the consistency of the overall system, the domain-specific models are integrated continuously with model transformation and synchronization techniques (cf. Sect. 5.1). By this means changes in one domain that are also relevant for other domains are communicated. The integrated results of the domains involved are tested with a virtual prototype. Failures and inconsistencies can be identified early and therefore cost and time consuming iterations in the development process can be avoided. Sect. 5.6 presents the approach of the CRC 614 regarding virtual prototyping and testing. Before start of production, a real prototype is built and tested. In case of satisfactory test results, the manufacturing documents are derived and the system is produced.

With the provided design methodology efficient self-optimizing systems can be developed that are able to adapt optimally to changing operation conditions and therefore meet todays requirements. The design methodology is extended by the aspect dependability in the book "Dependability of Self-optimizing Mechatronic Systems". The application examples of the CRC 614 show both the benefits and the transfer of the approaches to other areas. The CRC 614 has laid the foundation during its eleven years of research in the field of self-optimizing systems. Our future goal is to promote research in the area of intelligent technical systems as well as to bring these systems into use. For this purpose, the work of the CRC 614 will be continued in the leading-edge cluster “Intelligent Technical Systems Ost-Westfalen Lippe - Its OWL”.[18]

[18] For further information cf. www.its-owl.de

# Index

**A**

Active Guidance Module 28, 36, 160, 228
Active Suspension Module 28, 38, 160, 162, 231, 238, 248
Activity 126
Adaptation process 126
Adaptive systems 4
Air gap 33
  air gap adjustment 33
Algorithmic differentiation 227
AMS *see* Autonomous Mechatronic System
Autonomous Mechatronic System 8

**B**

Battery 42
Bayesian Network 168
  dictionary of translation rules 169
  translation algorithm 169
BeBot 50, 251
  grafter module 51
  lifter module 51
  transporter module 52
Behavior 16
Behavior planning 253
  hybrid planning architecture 254
  just-in-case planning 255
  online planning 255
  probabilistic planning model 253
Behavioral adaptation 7
  compositional adaptation 7
  parameter adaptation 7
  reconfiguration 7
  structure adaptation 7
BN *see* Bayesian Network

**C**

Capacitor 42
Challenge
  system complexity 4, 5
Chameleon, 225 227
Change
  domain-spanning relevant 187
Cognitive functions 139, 141–143, 147
  of information processsing 142
Component model 198
Component story diagram 203, 218
Conceptual design *see* Domain-spanning conceptual design, 162
  principle solution 67
Conditional Probability Table 169
CONSENS, *see* Specification technique CONSENS, *see* Specification technique CONSENS, *see* Specification technique CONSENS
Consistency analysis 129
Convoy 49
CPT, *see* Conditional Probability Table
Crosslinked Test Benches 46

**D**

Deductive analysis 135
Deployment 221
Design and development 67

(sub)system integration
virtual prototype 67
system model 67
consistency 67
Design Flow for Homogeneous Hard Macros 273
Design methodology 66
methods 66
solution pattern 66
reference process 66
tools 66
Design rules
application of 158
selection of 157
Design Structure Matrix 158, 175
aggregation DSM 158
Deterministic planning 17
Development task
analysis of 157
basic types of 154, 155
profile 154, 155
criteria for description of 154
Discrete behavior
consistency analysis of
correction of inconsistencies 133
interactive simulation 133
DMOC 15, 240, 245
Domain 66, 67, 74
advanced mathematics 66, 67
artificial intelligence 66, 67
control engineering 66, 74
electrical/electronic engineering 66, 74, 88
mechanical engineering 66, 74
software engineering 66, 74
Domain-spanning conceptual design 69
concept integration 69
conceptual design on the subsystem level 69
conceptual design on the system level 69
planning and clarifying the task 69, 70
principle solution 69
Domain-specific design and development 74, 76
(sub)system integration 74, 76, 91, 94
synchronization 91
system model 91
virtual prototype 91
control engineering 76, 78, 93, 94
electrical engineering 76, 78, 94
power electronic 88, 91
electronic engineering 76, 94
microelectronic devices 88
interaction 76, 93
mechanical engineering 76, 78, 93, 94
software engineering 76, 82, 93
discrete software 82, 83
system software 82, 85
tailoring 76
DSM *see* Design Structure Matrix

**E**

Economic efficiency 171
benefit 171, 178, 179
costs 171, 176, 178
Efficiency 33
Emulation 309
Energy Management 42, 46
Event 126
External objectives 124

**F**

Failure
cause 163
effect 163
mode 163
combinations 163
propagation
specification of 165
taxonomy 162, 164
Failure Mode and Effects Analysis 133, 136, 162, 163
Fault Tree *see* Fault Tree Analysis
Fault Tree Analysis 133, 135, 162, 165
Feasible set 230
FHA *see* Functional Hazard Analysis
Flexible Resource Manager 297
Flow specification 3
energy flow 3, 123
information flow 3, 123
material flow 3, 123
FMEA *see* Failure Mode and Effects Analysis
FO-TCTL 207
FTA *see* Fault Tree Analysis

Functional Hazard Analysis 162
Fussell-Vesely importance measure 168

G

GAIO 224
Graph transformation 190
Graph transformation system 220
  rule 220
  time 206

H

Hardware-in-the-Loop 28, 38, 277
Hierarchical Flexible Resource Manager 305
Hierarchical optimization 46
Hybrid Energy Storage 42, 229
Hybrid planning 18, 246
Hypervisor 292, 305

I

Incremental update 187
Inductive analysis 136
Influences
  distrubing 135
    deductive Analysis of 135
  disturbing 136
    inductive analysis of 136
  of the environment 120
Inherent objectives 125
Integrated circuits 227
Intelligent Drive Module 33
Intelligent technical systems 4
  adaptive 4
  foresighted 4
  robust 4
  user-friendly 4
Internal objectives 125

K

Kuhn-Tucker equations 13, 224, 230, 236, 238

L

Linear doubly-fed asynchronous drive 30
Linear drive 33
Linear motor 28, 231

M

Markov decision process 250
  partially observable 252
MATLAB
  Simulink 190, 220
  Stateflow 193
Mechatronic Function Module 8
Mechatronic Modeller 127
Mechatronic systems 2
  actuators 3
  basic structure 2
  basic system 2
  flow specifications, *see* Fow specification3
  information processing 2
  sensors 2
Mechatronics *see* Mchatronic systems2
MechatronicUML 83, 188, 197
  MechatronicUML process, 83
MFM *see* Mechatronic Function Module
MIM *see* Module Indication Matrix
Modal Sequence Diagrams (MSD) 130, 199
  Assumption MSDs 130
  lifeline 130
  messages 131
    cold messages 131
    hot messages 131
  order of messages 132
  Requirement MSDs 130
  specification 132
  syntax of 130
  violation 132
    cold violation 132
    hot violation 132
Model
  domain-spanning model 186
  domain-specific model 186
  hierarchical 232
Model synchronization 186, 194
Model transformation 186
  bidirectional 194
  Triple Graph Grammar 190
Model-based optimization 12
Model-order reduction
  parametric 232

projection-based 234
Module 155
Module Indication Matrix 160
Motion function 8
Motion planning 244
with motion primitives 244
MSD, *see* Modal Sequence Diagrams
Multi Processor System on Chip, 267
Multicriteria optimization see Multiobjective optimization13
Multiobjective optimal control problem 15, 241
Multiobjective optimization 12, 13, 35, 40, 60, 224, 232
problem, 13
recovering algorithm 225
set-oriented methods 14, 224
subdivision algorithm 225
decision making 15
hierarchical 229
parametric 236–238
Pareto front 227
Pareto set 224, 225, 227, 228
problem 224, 237
bilevel 230
recovering algorithm 225, 227, 228
subdivision algorithm 227
Multiple Domain Matrix 174

**N**

Network on-chip 267
Networked Mechatronic System 9
NMS *see* Networked Mechatronic System
Norm
IEC 60812 163
CENELEC EN 50129 162

**O**

Objective function 40
Objective Priority Matrix 140, 141
Objective-based controller 40
Objectives 139
external 6
inherent 6
internal 6
Observer 295
Observer-Controller Architecture 294
OCM *see* Operator-Controller-Module, *see* Operator-Controller-Module
OMEGA 68
business process 68
connector 69
exclusives or decisions 69
process objects 69
synchronization line 69
Operating point control 29
Operating strategy 229
Operation process 126
Operator-Controller-Module 10, 231, 232
Cognitive Operator 10, 150
Controller 10, 148
Reflective Operator 10, 148
Optimal control 14, 239
of electrical systems 242
of hybrid systems 243
of mechanical systems 240
of mechatronic systems 242
direct methods 15, 240
problem 12, 14, 228, 239
Optimization 58

**P**

PALMERA 267
Paravirtualization 306
Pareto front 13, 35
Pareto point 13
robust 237
Pareto set 13, 15, 36, 41, 225, 233
parametrization 230
Partial models 119
active structure 119, 123
application scenarios 119, 120, 129
consistency analysis of 129
formal description of 129
behavior, 119 126
behavior–activities 126
behavior–sequence 126
behavior–states 126
coherent system of, *see* Principle solution
environment 119
functions 119, 123
interrelations 126, 127
objectives
conflict between 125, 133, 137
hierarchy of 134

requirements 119, 121
demands 122
wishes 122
shape 119, 125
system of objectives 119, 124, 133
design of 133
external objectives 124
inherent objectives 125
internal objectives 125
Path following
numerical 236, 237
Planning
continuous 246
discrete 246
hybrid 246
offline 247
Planning Domain Definition Language 220
Policy 250
Posterior probability 170
Principle solution 118, 119, 162, 186
aspects of the description of *see* Partial models
data model of 128
Probabilistic planning 18
Probabilistic reliability analysis 162
Product structure 158

**Q**

Q-Learning 251

**R**

RailCab 27, 38, 228, 231, 237
active structure of 124
application scenarios 121
application scenarios for 130
environment of 120
external ojectives of 125
failure specification of 169
function hierarchy of 123
inherent objectives of 125
internal objectives of 125
Modal Sequence Diagrams of 132
modules of 160
objectives of 125
potential conflicts 138
product structure of 161, 163
function-oriented 160
shape-oriented 160
requirements on 122
Rapid Prototyping Platfrom RAPTOR 280
Real-time 293
Real-time coordination pattern 200
multirole 201
role 201
singlerole 201
subrole 201
Real-time Operating System 293, 299
Real-time statechart 201, 205
Reconfiguration 19, 294
hardware 21
self-optimizing application 20
system software 21
Reconfiguration Structure Matrix 158
Redundancy 169
Reference process
domain-spanning conceptual design 66
domain-specific design and development 67
Refinement 186
real-time coordination pattern 210
Reliability 162
Requirements
traceability 127, 129
Reward 250
Risk Priority Number 163
RoSM, *see* Reconfiguration Structure Matrix
RPN, *see* Risk Priority Number
Runtime reconfiguration 218, 219

**S**

Safe planning 219
Schedulability 298
Self-optimization 5
key aspects 6
system of objectives 6
Self-optimization process 6, 144
Self-optimizing control structure 38
Self-optimizing development process 96
Self-optimizing Operating Point Control 30
Self-optimizing systems 5
product structure of
function-oriented 160

shape-oriented 160
validation of 160
with regard to additional aspects 160
with regard to flow interdependencies 158
with regard to flow reconfiguration 159
with regard to spatial interdependencies 158
product structuring of 154
design rules for 155
example of a design rule 156
system dynamics 174, 175
Solution pattern 143, 144, 146, 147, 150, 261
aspects 144
active Structure 152
active structure 144
behavior 145, 152
characteristics 144
context 146
functions 144
implementation 145
for the basic system 144
prohabilistic planning 150, 152
tool support 146
uniform specification of 145
Specification technique CONSENS 119
abstract syntax of 128
concrete syntax of 129
dynamic semantics of 129
metamodel of 128
software support for *see* Mechatronic Modeller
static semantics of 128
State of the system 126
State transition 126
Statistical planning 250
Switched Reluctance Motor 32, 242
Synthesis 215
SysML *see* Systems Modeling Language
System element 124
System virtualization 305
Systems Modeling Language 126

**T**

Test rig 33, 38
Test track 28
TGG *see* Triple Graph Grammar
Trade-off between objectives see Multiobjective optimization 13
Transition *see* State transition
Triple Graph Grammar 190

**U**

UML *see* Unified Modeling Language
Unified Modeling Language 126

**V**

Vehicle dynamics 57, 58
Vehicle-dynamics control 60
Verification
design-time 204, 220
Virtual Environment 310, 311, 314, 336
Virtual machine 305
migration 309
Virtual Prototype 310–312, 314
software agent 314
Virtual Prototype Agent 321, 322
Virtual Test Bench 311, 312, 331, 332, 335, 336
mutant 332–335
simulation 335
self-optimizing 332, 334
Virtualization 305
Visualization Agents 320–322

**X**

X-by-wire vehicle 56