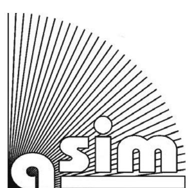
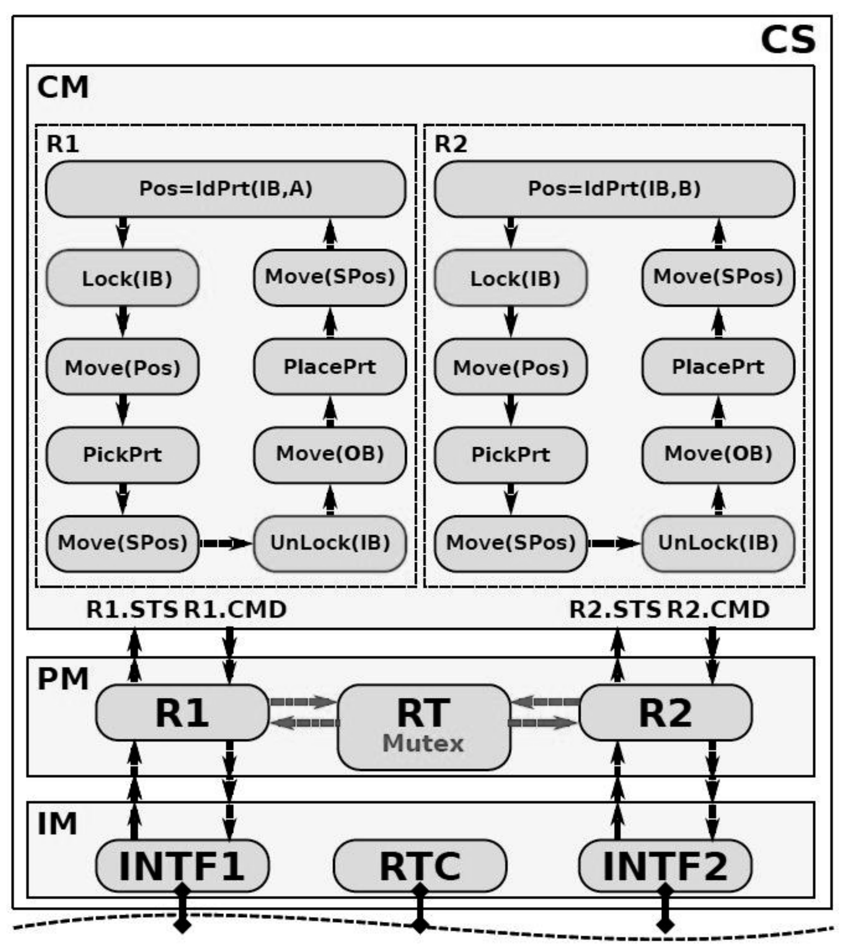


FBS 40

Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS

Birger Freymann



Fortschrittsberichte Simulation
Advances in Simulation

ISBN print 978-3-903311-20-6 ISBN ebook 978-3-903347-40-3 DOI 10.11128/fbs.40

ASIM Books – ASIM Book Series – ASIM Buchreihen

Proceedings

Proceedings Langbeiträge ASIM SST 2022 -26. ASIM Symposium Simulationstechnik, TU Wien, Juli 2022
F. Breitenecker, C. Deatcu, U. Durak, A. Körner, T. Pawletta (Hrsg.), ARGESIM Report 20; ASIM Mitteilung AM 181
ISBN ebook 978-3-901608-97-1, DOI 10.11128/arep.20, ARGESIM Verlag Wien, 2022; ISBN print 978-3-903311-19-0, TU Verlag

Proceedings Kurzbeiträge ASIM SST 2022 -26. ASIM Symposium Simulationstechnik, TU Wien, Juli 2022
F. Breitenecker, C. Deatcu, U. Durak, A. Körner, T. Pawletta (Hrsg.), ARGESIM Report 19; ASIM Mitteilung AM 179
ISBN ebook 978-3-901608-96-4, DOI 10.11128/arep.19, ISBN print 978-3-901608-73-5, ARGESIM Verlag Wien, 2022

Simulation in Production and Logistics 2021 – 19. ASIM Fachtagung Simulation in Produktion und Logistik
Online Tagung, Sept. 2021, J. Franke, P. Schuderer (Hrsg.), Cuvillier Verlag, Göttingen, 2021,
ISBN print 978-3-73697-479-1; ISBN ebook 978-3-73696-479-2; ASIM Mitteilung AM177

Proceedings ASIM SST 2020 – 25. ASIM Symposium Simulationstechnik, Online-Tagung
14.-15.10.2020; C. Deatcu, D. Lückerath, O. Ullrich, U. Durak (Hrsg.), ARGESIM Verlag Wien, 2020;
ISBN ebook: 978-3-901608-93-3; DOI 10.11128/arep.59; ARGESIM Report 59; ASIM Mitteilung AM 174

Simulation in Production and Logistics 2019 – 18. ASIM Fachtagung Simulation in Produktion und Logistik
Chemnitz, 18.-20. 9. 2019; M. Putz, A. Schlegel (Hrsg.), Verlag Wissenschaftliche Skripten Auerbach, 2019,
ISBN print 978-3-95735-113-5, ISBN ebook 978-3-95735-114-2; ASIM Mitteilung AM172

Tagungsband ASIM SST 2018 - 24. ASIM Symposium Simulationstechnik, HCU Hamburg, Oktober 2018
C. Deatcu, T. Schramm, K. Zobel (Hrsg.), ARGESIM Verlag Wien, 2018; ISBN print: 978-3-901608-12-4;
ISBN ebook: 978-3-901608-17-9; DOI 10.11128/arep.56; ARGESIM Report 56; ASIM Mitteilung AM 168

Book Series Fortschrittsberichte Simulation – Advances in Simulation

Cooperative and Multirate Simulation: Analysis, Classification and New Hierarchical Approaches. I. Hafner, FBS 39
ISBN ebook978-3-903347-39-7, DOI 10.11128/fbs.39, ARGESIM Publ. Vienna,2022; ISBN print978-3-903311-07-7, TUVerlag Wien, 2022

Die Bedeutung der Risikoanalyse für den Rechtsschutz bei automatisierten Verwaltungsstrafverfahren. T. Preiß, FBS 38
ISBN ebook 978-3-903347-38-0, DOI 10.11128/fbs.38, ARGESIM Publ. Vienna,2020; ISBN print 978-3-903311-14-5, TUVerlag Wien, 2020

Methods for Hybrid Modeling and Simulation-Based Optimization in Energy-Aware Production Planning. B. Heinzl, FBS 37
ISBN ebook 978-3-903347-37-3, DOI 10.11128/fbs.37, ARGESIM Publ. Vienna,2020; ISBN print 978-3-903311-11-4, TUVerlag Wien, 2020

Konforme Abbildungen zur Simulation von Modellen mit verteilten Parametern. Martin Holzinger, FBS 36
ISBN ebook 978-3-903347-36-6, DOI 10.11128/fbs.36, ARGESIM Publ. Vienna, 2020; ISBN print 978-3-903311-10-7, TUVerlag Wien, 2020

Fractional Diffusion by Random Walks on Hierarchical and Fractal Topological Structures. G. Schneckenreither, FBS 35
ISBN ebook 978-3-903347-35-9, DOI 10.11128/fbs.35, ARGESIM Publ. Vienna, 2020

A Framework Including Artificial Neural Networks in Modelling Hybrid Dynamical Systems. Stefanie Winkler, FBS 34
ISBN ebook 978-3-903347-34-2, DOI 10.11128/fbs.34, ARGESIM Publ. Vienna, 2020; ISBN print 978-3-903311-09-1, TUVerlag Wien, 2020

Modelling Synthesis of Lattice Gas Cellular Automata and Random Walk and Application to Gluing of Bulk Material. C. Rößler, FBS 33
ISBN ebook 978-3-903347-33-5, DOI 10.11128/fbs.33, ARGESIM Publ. Vienna, 2020; ISBN print 978-3-903311-08-4, TUVerlag Wien, 2020

Combined Models of Pulse Wave and ECG Analysis for Risk Prediction in End-stage Renal Desease Patients. S. Hagmair, FBS 32
ISBN ebook 978-3-903347-32-8, DOI 10.11128/fbs.32, ARGESIM Publ. Vienna, 2020

Mathematical Models for Pulse Wave Analysis Considering Ventriculo-arterial Coupling in Systolic Heart Failure. S. Parragh, FBS 31
ISBN ebook 978-3-903347-31-1, DOI 10.11128/fbs.31, ARGESIM Publ. Vienna, 2020

Variantenmanagement in der Modellbildung und Simulation unter Verwendung des SES/MB Frameworks. A. Schmidt, FBS 30;
ISBN ebook 978-3-903347-30-4, DOI 10.11128/fbs.30, ARGESIM Verlag, Wien 2019; ISBN print 978-3-903311-03-9, TUVerlag Wien, 2019

Classification of Microscopic Models with Respect to Aggregated System Behaviour. Martin Bicher, FBS 29;
ISBN ebook 978-3-903347-29-8, DOI 10.11128/fbs.29, ARGESIM Publ. Vienna, 2017; ISBN print 978-3-903311-00-8, TUVerlag Wien, 2019

Model Based Methods for Early Diagnosis of Cardiovascular Diseases. Martin Bachler, FBS 28;
ISBN ebook 978-3-903347-28-1, DOI 10.11128/fbs.28, ARGESIM Publ. Vienna, 2017; ISBN print 978-3-903024-99-1, TUVerlag Wien, 2019

A Mathematical Characterisation of State Events in Hybrid Modelling. Andreas Körner, FBS 27;
ISBN ebook 978-3-903347-27-4, DOI 10.11128/fbs.27, ARGESIM Publ. Vienna, 2016

Comparative Modelling and Simulation: A Concept for Modular Modelling and Hybrid Simulation of Complex Systems. N.Popper,
FBS 26; ISBN ebook 978-3-903347-26-7, DOI 10.11128/fbs.26, ARGESIM Publ. Vienna, 2016

Rapid Control Prototyping komplexer und flexibler Robotersteuerungen auf Basis des SBE-Ansatzes. Gunnar Maletzki, FBS 25;
ISBN ebook 978-3-903347-25-0, DOI 10.11128/fbs.25, ARGESIM Publ. Vienna, 2019; ISBN Print 978-3-903311-02-2, TUVerlag Wien, 2019

A Comparative Analysis of System Dynamics and Agent-Based Modelling for Health Care Reimbursement Systems. P. Einzinger,
FBS 24; ISBN ebook 978-3-903347-24-3, DOI 10.11128/fbs.24, ARGESIM Publ. Vienna, 2016

Agentenbasierte Simulation von Personenströmen mit unterschiedlichen Charakteristiken. Martin Bruckner, FBS 23;
ISBN ebook Online 978-3-903347-23-6, DOI 10.11128/fbs.23, ARGESIM Verlag Wien, 2016

Deployment of Mathematical Simulation Models for Space Management. Stefan Emrich, FBS 22;
ISBN ebook 978-3-903347-22-9, DOI 10.11128/fbs.22, ARGESIM Publisher Vienna, 2016

Lattice Boltzmann Modeling and Simulation of Incompressible Flows in Distensible Tubes for Applications in Hemodynamics.
X. Descovich, FBS 21; ISBN ebook 978-3-903347-21-2, DOI 10.11128/fbs.21, ARGESIM, 2016; ISBN Print 978-3-903024-98-4, TUVerlag 2019

Mathematical Modeling for New Insights into Epidemics by Herd Immunity and Serotype Shift. Florian Miksch, FBS 20;
ISBN ebook 978-3-903347-20-5, DOI 10.11128/fbs.20, ARGESIM Publ. Vienna, 2016; ISBN Print 978-3-903024-21-2, TUVerlag Wien, 2016

Integration of Agent Based Modelling in DEVS for Utilisation Analysis: The MoreSpace Project at TU Vienna. S.Tauböck, FBS 19
ISBN ebook 978-3-903347-19-9, DOI 10.11128/fbs.19, ARGESIM Publ., 2016; ISBN Print 978-3-903024-85-4, TUVerlag Wien, 2019

Download via ASIM - **www.asim-gi.org** **Print-on-Demand** via TUVerlag **www.tuverlag.at**

asim
ARGESIM

FBS Fortschrittsberichte Simulation
Advances in Simulation

Herausgegeben von **ASIM - A**rbeitsgemeinschaft **Sim**ulation, Fachausschuss der **GI** – Gesellschaft für Informatik - im Fachbereich **ILW** – Informatik in den Lebenswissenschaften
Published by **ASIM** – German Simulation Society, Section of **GI** – German Society for Informatics i-in Division **ILW** – Informatics in Life Sciences

ASIM FBS 40

Birger Freymann

Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS

FBS - Fortschrittsberichte Simulation / Advances in Simulation

Herausgegeben von **ASIM** - Arbeitsgemeinschaft Simulation, → *www.asim-gi.org*

ASIM ist ein Fachausschuss der **GI** - Gesellschaft für Informatik, → *www.gi.de*, im Fachbereich **ILW** – Informatik in den Lebenswissenschaften, → *fb-ilw.gi.de*

Published on behalf of **ASIM** – German Simulation Society, → *www.asim-gi.org*

ASIM is a section of of **GI** – German Society for Informatics, → *www.gi.de*, in division **ILW** – Informatics in the Life Sciences, → *fb-ilw.gi.de*

Reihenherausgeber / Series Editors

Prof. Dr.-Ing. Th. Pawletta (ASIM), HS Wismar, Thorsten.pawletta@hs-wismar.de

Prof. Dr. D. Murray-Smith (EUROSIM / ASIM), Univ. Glasgow,
David.Murray-Smith@glasgow.ac.uk

Prof. Dr. F. Breitenecker (ARGESIM / ASIM), TU Wien, Felix.Breitenecker@tuwien.ac.at

Titel / Title: Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS

Autor /Author: Birger Freymann, *Birger.Freymann@hs-wismar.de*

FBS Band / Volume: 40

Typ / Type: Dissertation

ISBN print: 978-3-903311-20-6, TU-Verlag, Vienna, 2022;
Print-on-Demand, www.tuverlag.at

ISBN ebook: 978-3-903347-40-3, ARGESIM Publisher Vienna, 2022;
www.argesim.org

DOI: 10.11128/fbs.40

Seiten / Pages: 168 + xii

© by ARGESIM / ASIM, Wien, 2022

Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS

Dissertation

zur Erlangung des Doktorgrades
der Ingenieurwissenschaften

vorgelegt von
Birger Freymann
aus Ludwigslust

genehmigt von der
Fakultät für Mathematik/Informatik und Maschinenbau
der Technischen Universität Clausthal,

Tag der mündlichen Prüfung
06.04.2022

Dekan
Prof. Dr. Jörg P. Müller

Vorsitzender der Promotionskommission

Betreuer
Prof. Dr. Sven Hartmann

Gutachter
apl. Prof. Dr.-Ing. Umut Durak

Gutachter
Prof. Dr.-Ing. Thorsten Pawletta, HS Wismar

Danksagung

Die vorliegende Arbeit ist während meiner Tätigkeit an der Hochschule Wismar in der Forschungsgruppe Computational Engineering und Automation im Rahmen einer Kooperation mit der Technischen Universität Clausthal entstanden.

Mein aufrichtiger Dank gilt meinen Doktorvätern Prof. Dr.-Ing. Thorsten Pawletta und Prof. Dr.-Ing. Sven Pawletta für die Bereitstellung des interessanten Themas, sowie für die Betreuung und stetige Unterstützung während meiner Arbeit. Mein besonderer Dank gilt dabei Thorsten Pawletta, der viel Zeit für mich opferte und mich durch seine konstruktiven Ratschläge entscheidend vorangebracht hat.

Besonders danke ich Herrn Prof. Dr. Sven Hartmann, für die Bereitschaft zur Übernahme der Betreuung von Seiten der TU-Clausthal. Mein gleicher Dank geht an Prof. Dr. Umut Durak von der TU-Clausthal / DLR Braunschweig für wertvolle Hinweise während der Bearbeitung des Promotionsvorhabens und die Bereitschaft zur Begutachtung.

Auch ein ganz großes Dankeschön an alle Mitarbeiter der Forschungsgruppe Computational Engineering und Automation für die freundliche Zusammenarbeit, die schöne Zeit und Hilfe bei allen kleinen und größeren Problemen.

Nicht vergessen möchte ich dieser Stelle meine Familie. Meinen Eltern danke ich für die Förderung und Unterstützung meines Studiums. Meiner Schwester Undine danke ich für die vielen Hinweise und konstruktiven Gespräche.

Kurzfassung

In der Industrie sind Roboter bereits seit Jahrzehnten als leistungsfähige und flexible Werkzeuge etabliert. Mit neuen Anwendungsbereichen und Anforderungen, wie sie zum Beispiel im Rahmen der *Industrie 4.0 Initiative* definiert werden, kommt der effizienten Entwicklung von Robotersteuerungen eine immer größere Bedeutung zu. Diese Tendenz erfordert die Entwicklung neuer Methoden zur herstellerunabhängigen und applikationsübergreifenden Programmierung von Roboteranwendungen. Entscheidend sind hierbei systematische Vorgehensmodelle, theoriekonforme und modellbasierte Entwicklungsmethoden sowie moderne Programmiersysteme als Entwicklungswerkzeuge. Die Vielzahl der herstellerspezifischen Entwicklungsumgebungen auf dem Markt zeigt, dass sich existierende Standards im Bereich der Steuerungsentwicklung für Roboter bisher kaum durchsetzen konnten. Die methodische und softwaretechnische Diversität erschwert es, unterschiedliche Robotertypen und Roboter unterschiedlicher Hersteller in Teams zu gruppieren, um leistungsfähige, flexible und kostengünstige Multi-Robotersysteme (MRS) umzusetzen. Durchgängige Toolketten sowie Methoden der Modellbildung und Simulation spielen dabei eine wichtige Rolle. In der Literatur wird in diesem Zusammenhang vom Rapid-Control-Prototyping (RCP) gesprochen. Gegenwärtig sind RCP-basierte Techniken in der industriellen Robotik fast immer herstellerspezifisch.

In dieser Arbeit wird ein Ansatz zur durchgängigen, modellbasierten und herstellerunabhängigen Steuerungsentwicklung für Roboterteams mit industriellen Knickarmrobotern entwickelt. Aufbauend auf dem Simulation-Based-Control (SBC)-Ansatz und dem Task-Oriented-Control (TOC)-Ansatz werden Entwicklungsmethoden aus dem Bereich von Single-Robotersystemen (SRS) auf MRS übertragen. Es werden mögliche Interaktionen zwischen Robotern untersucht und hierauf aufbauend Interaktionsklassen definiert. Zur Umsetzung einer durchgängigen Steuerungsentwicklung wird der Discrete-Event-System-Specification (DEVS)-Formalismus diskutiert und es werden Erweiterungen zur Echtzeit- und Prozessanbindung untersucht. Hieraus abgeleitet wird ein modifizierter Formalismus entwickelt und dessen Eignung zur durchgängigen, modellbasierten Steuerungsentwicklung anhand eines Fallbeispiels demonstriert. Zur Spezifikation der entwickelten Modelle wird die DEVS-Diagramm-Notation verwendet und um zusätzliche Beschreibungsmittel erweitert. Basierend auf den zuvor definierten Interaktionsklassen werden Lösungsansätze zur TOC-basierten Umsetzung von Interaktionen mittels modularer und wiederverwendbarer Aufgaben erarbeitet. Anschließend wird deren prototypische Umsetzung anhand von Fallbeispielen mittels des neu entwickelten DEVS-Formalismus gezeigt. Die Komplexität der Interaktionen steigt mit jedem Fallbeispiel an. Zur besseren Handhabung der Komplexität wird mit dem Erweiterten System-Entity-Structure/Model-Base (SES/MB)-Ansatz eine zusätzliche modellbasierte Technik eingeführt, mit dem SBC- sowie dem TOC-Ansatz integriert und anhand eines Fallbeispiels der bisherigen Vorgehensweise gegenübergestellt.

Abstract

Robots have been established in industry for decades as powerful and flexible tools. With new application areas and requirements, such as those defined in the context of the Industry 4.0 initiative, the efficient development of robot controllers is becoming increasingly important. This trend requires the development of new methods for the manufacturer-independent and application-independent programming of robot applications. The decisive factors here are systematic procedural models, theory-compliant and model-based development methods, and modern programming systems as development tools. The large number of manufacturer-specific development environments on the market shows that existing standards in the area of control development for robots have hardly been able to establish themselves. The methodological and software diversity makes it difficult to group different robot types and robots from different manufacturers in teams in order to implement powerful, flexible and cost-effective multi-robot systems (MRS). Continuous tool chains as well as methods of modeling and simulation play an important role in this context. In the literature, rapid control prototyping (RCP) is spoken of in this context. Currently, RCP-based techniques in industrial robotics are almost always manufacturer-specific.

In this thesis, an approach for end-to-end, model-based and vendor-independent control development for robot teams with industrial jointed-arm robots is developed. Based on the Simulation-Based-Control (SBC) approach and the Task-Oriented-Control (TOC) approach, development methods from the field of single robot systems (SRS) are transferred to MRS. Possible interactions between robots are investigated and, based on this, interaction classes are defined. For the implementation of an integrated control development the Discrete-Event-System-Specification (DEVS) formalism is discussed and extensions for real-time and process integration are investigated. Derived from this, a modified formalism is developed and its suitability for integrated, model-based control development is demonstrated by means of a case study. For the specification of the developed models the DEVS diagram notation is used and extended by additional descriptive means. Based on the previously defined interaction classes, approaches for the TOC-based implementation of interactions are developed using modular and reusable tasks. Subsequently, their prototypical implementation is shown on the basis of case studies using the newly developed DEVS formalism. The complexity of the interactions increases with each case study. For a better handling of the complexity, the Extended System-Entity-Structure/Model-Base (SES/MB) approach is introduced as an additional model-based technique, integrated with the SBC and TOC approaches, and compared with the previous approach using a case study.

Inhaltsverzeichnis

Abkürzungsverzeichnis xi

1 Einleitung 1
1.1 Motivation . . . 1
1.2 Background . . . 2
1.2.1 Funktionsweise von Gelenkarmrobotern . . . 2
1.2.2 Roboterorientierte Middleware . . . 3
1.2.3 Online vs. Offline-Programmierung . . . 4
1.2.4 Systematische Entwicklung und Aufgabenorientierung . . . 4
1.2.5 Interaktionen in der Robotik . . . 5
1.2.6 Formale Methoden und Model-based Design . . . 6
1.3 Zielsetzung, Forschungshypothesen und Struktur der Arbeit . . . 7

2 Entwurf, Programmierung und Inbetriebnahme von Robotersteuerungen 11
2.1 Steuerungsarchitektur . . . 11
2.1.1 Steuerungsparadigma und Aktivitätsniveau . . . 12
2.1.2 Roboterorientierte-Middleware . . . 16
2.2 Methoden der Roboterprogrammierung . . . 18
2.2.1 Online-Methoden . . . 19
2.2.2 Offline-Methoden . . . 20
2.3 Vorgehensmodelle und Frameworks . . . 23
2.3.1 V-Modell . . . 24
2.3.2 Rapid-Control-Prototyping (RCP) . . . 26
2.3.3 Simulation-Based-Control (SBC) Ansatz . . . 27
2.4 Aufgabenorientierte Steuerungen (TOC) mit dem SBC-Ansatz . . . 28
2.4.1 Grundprinzip und Schichten einer TOC . . . 29
2.4.2 Umsetzung von TOC im SBC-Ansatz . . . 30
2.5 Interaktionen in Multi-Robotersystemen (MRS) . . . 31
2.5.1 Charakteristik von Roboterteams . . . 32
2.5.2 Interaktionsklassen von Roboterteams . . . 33
2.6 Zusammenfassung . . . 37

3 Ereignisdiskrete Simulation und Steuerung mit dem DEVS-Formalismus 39
3.1 Beziehung zwischen ereignisdiskreter Simulation und ereignisdiskreter Steuerung . . . 40
3.2 DEVS-Formalismen . . . 40
3.2.1 Classic-DEVS-Formalismus und die Erweiterung DEVS mit Ports . 41
3.2.2 Parallel-DEVS-Formalismus . . . 45

3.2.3 DEVS-Diagramm-Notation und eingeführte Erweiterungen 48
3.3 Echtzeit- und Prozessanbindung 51
3.3.1 Real-Time-DEVS-Formalismus 51
3.3.2 Action-Level-Real-Time-DEVS 57
3.3.3 Real-Time-Embedded-DEVS 60
3.3.4 DEVS-Based Transparent M&S Framework 62
3.3.5 PDEVS-RCP-Formalismus 64
3.4 Zusammenfassung 67

4 Aufgabenorientierte Steuerungen für MRS mit SBC und DEVS 69
4.1 DEVS im Kontext der durchgängigen Steuerungsentwicklung 69
4.1.1 Probleme bestehender DEVS-Ansätze 70
4.1.2 Entwicklung des PDEVS-RCP-V2-Formalismus 71
4.2 Beispiel zum SBC mit PDEVS-RCP-V2 75
4.2.1 Übergang von der Entwurfs- zur Automatisierungsphase 76
4.2.2 Übergang von der Automatisierungsphase zum operativen Betrieb mit PDEVS-RCP 78
4.2.3 Übergang von der Automatisierungsphase zum operativen Betrieb mit PDEVS-RCP-V2 78
4.2.4 Aufgabenorientierte Betrachtung 79
4.3 TOC basierte Lösungsansätze für MRS 80
4.3.1 Interaktionsklasse 1 und 2 80
4.3.2 Interaktionsklasse 3 82
4.3.3 Interaktionsklasse 4 85
4.3.4 Interaktionsklasse 5 88
4.3.5 Interaktionsklasse 6 89
4.4 Zusammenfassung 91

5 Flexible aufgabenorientierte Multi-Robotersteuerungen 93
5.1 System-Entity-Structure/Model-Base Framework 94
5.2 Flexible Steuerungen auf Basis des Erweiterten SES/MB-Frameworks . . . 98
5.3 Zusammenfassung 100

6 Implementierung von TOC für MRS 101
6.1 MATLAB als SBC-Plattform 101
6.1.1 Auswahl einer Roboter-Middleware 102
6.1.2 Umsetzung eines echtzeitfähigen DEVS-Formalismus 103
6.1.3 Umsetzung des SES/MB-Ansatzes 104
6.1.4 Umsetzung des Erweiterten SES/MB-Frameworks 106
6.2 Umsetzung ausgewählter Lösungsansätze 107
6.2.1 Interaktionsklasse 1 und 2 108
6.2.2 Interaktionsklasse 3 113
6.2.3 Interaktionsklasse 6 116
6.2.4 Interaktionsklasse 6 mit Erweitertem SES/MB-Framework 120
6.3 Zusammenfassung 126

7 Zusammenfassung und Ausblick auf weiterführende Arbeiten 127

Literaturverzeichnis 130

Abbildungsverzeichnis **145**

Tabellenverzeichnis **149**

Listings **151**

Anhang **153**

A Diskussion zur Umsetzung der Komponente RTC 153
B Modellspezifikationen zum Abschnitt 4.2 154
C Lösungsansatz zu Interaktionsklasse 6 160
D Axiome der System-Entity-Structure (SES) 161
E MATLAB/DEVS Quellcode zu Kapitel 6 161

Abkürzungsverzeichnis

ACC Acceptor
ALRT-DEVS Action-Level-Real-Time-DEVS
CAD Computer-Aided-Design
CAR Computer-Aided-Robotics
CM Control-Model
CPS Cyber-Physical-System
CS Control-Software
DES Discrete-Event-Simulation
DEVS Discrete-Event-System-Specification
EC Execution-Control
EF Experimental-Frame
EU Execution-Unit
FNSS Function-Specified-System
FPES Flatted-Pruned-Entity-Structure
GEN Generator
HiL Hardware-in-the-Loop
IB Input-Buffer
IM Interfacemodell
IRL Industrial Robot Language
KMU kleine und mittelständische Unternehmen
KRL KUKA-Robotic-Language
MAS Multi-Agentensystem
MB Modellbibliothek
MRS Multi-Robotersystem
MUS Model-Under-Study

M&S	Modellbildung und Simulation
OB	Output-Buffer
OC	Overall-Control
PDEVS	Parallel-DEVS
PES	Pruned-Entity-Structure
PM	Prozessmodell
PROC	Prozessor
RCP	Rapid-Control-Prototyping
RMW	roboterorientierte-Middleware
ROS	Robot-Operating-System
RTC	Real-Time-Clock
RT-DEVS	Real-Time-DEVS
RT-DEVN	coupled-Real-Time-DEVS
RT-DM	Real-Time-Driver-Model
TR-DEVS	DEVS-Based-Transparent-Modeling-and-Simulation-Framework
RTE-PDEVS	Real-Time-Embedded-PDEVS
SBC	Simulation-Based-Control
SCE	Scientific-Computing-Environment
SES	System-Entity-Structure
SES/MB	System-Entity-Structure/Model-Base
SiL	Software-in-the-Loop
SM	Simulationsmodell
SRS	Single-Robotersystem
TOC	Task-Oriented-Control
TRA	Transducer
VT	Virtual-Time
VS	Verteiltes System
WCT	Wall-Clock-Time

1 Einleitung

Roboter sind in der Industrie seit Jahrzehnten als vielseitige Werkzeuge etabliert, da sie eine hohe Leistungsfähigkeit und Adaptivität aufweisen. Stetig werden neue Anwendungsfelder erschlossen, woraus erweiterte Anforderungen bezüglich der Konstruktion und Steuerungsentwicklung resultieren. Eine besondere Herausforderung für die Steuerungsentwicklung stellen Multi-Robotersysteme dar, die unterschiedliche Robotertypen oder Roboter unterschiedlicher Hersteller integrieren, sowie der Einsatz von Robotern in flexiblen Fertigungssystemen gemäß den Anforderungen der Industrie 4.0 (Hirsch-Kreinsen und Karačić [47]). In diesem Kontext kommt der effizienten Entwicklung von Robotersteuerungen eine immer stärkere Bedeutung zu, da die Softwarekosten in hohem Maße die Investitions- und Betriebskosten beeinflussen. Demzufolge ist das Ziel jeder Steuerungsimplementierung, dass der Entwurf und die Inbetriebnahme der Steuerung möglichst einfach, sicher, schnell und damit kostengünstig realisiert werden kann. Um diesen Anforderungen gerecht zu werden ist es wichtig, im Entwicklungsprozess Reimplementierungen zu vermeiden, modulare Teillösungen zu entwickeln und diese systematisch vor der Inbetriebnahme zu testen. Weiterhin sollten Robotersteuerungen möglichst herstellerunabhängig entwickelt werden, so dass Teams aus Robotern unterschiedlicher Hersteller effizient und sicher in Betrieb genommen werden können und der Wartungsaufwand für Steuerungen möglichst gering gehalten wird.

1.1 Motivation

Um komplexe Robotersteuerungen effizient zu implementieren, sollte die Entwicklung auf Basis von Vorgehensmodellen und Frameworks erfolgen. Entsprechende Ansätze hierfür liefert die Systemtheorie (Pichler und Schwaertzel [87]), das Control-Engineering (Lunze [60], Litz und Frey [58]) und das Software-Engineering (Balzert [6]). Im Kontext der Steuerungsentwicklung komplexer Systeme sind formale Methoden der Modellbildung und Simulation (M&S) sowie abstrahierte Programmiermethoden von Bedeutung. Bemühungen hinsichtlich standardisierter Entwicklungsmethoden oder interoperabler Programmiersprachen werden seitens der Roboterhersteller in der Regel kaum unterstützt. Dadurch lassen sich die in proprietären Entwicklungsumgebungen entwickelten Steuerungslösungen nicht auf Robotersysteme anderer Hersteller übertragen und die Steuerungsentwicklung für Multi-Robotersysteme (MRS) mit Robotern unterschiedlicher Hersteller wird zusätzlich erschwert. Aufgrund unterschiedlicher Kosten und Spezialisierungen seitens der Hersteller gibt es bei MRS, auch wenn diese nur aus Gelenkarmrobotern bestehen, mannigfaltige Gründe, um Roboter unterschiedlicher Hersteller in ein Team zu integrieren.

Diese Arbeit hat das Ziel, einen Beitrag zur durchgängigen und herstellerunabhängigen

Entwicklung von Robotersteuerungen für MRS zu leisten, wobei eine Beschränkung auf Teams von Gelenkarmrobotern erfolgt. Der gesamte Entwicklungsprozess soll auf formalen Methoden der M&S sowie abstrahierten Programmieransätzen aufgebaut werden. Mit Blick auf einen durchgängigen Entwicklungsprozess und das frühzeitige systematische Testen von Lösungen, soll insbesondere auf die Strukturierung und sukzessive Erweiterung von Steuerungsmodellen bis hin zu Steuerungsprogrammen im operativen Betrieb eingegangen werden. Hierzu wird auf ein herstellerunabhängiges Vorgehensmodell und Framework aufgebaut, welches einen durchgängigen Entwicklungsprozess und ein sukzessives Testen von Lösungen ermöglicht.

Eine systematische Entwicklung (Lüth und Längle [62]) von Multi-Robotersteuerungen erfordert, ergänzend zu den Anforderungen bei Single-Robotersteuerungen, eine Analyse und Klassifikation von Roboterinteraktionen sowie Softwarekonzepte zu deren programmtechnischen Umsetzung. Anhand von Interaktionsklassen soll gezeigt werden, dass sich Interaktionen modellbasiert unter Wiederverwendung und Komposition vordefinierter Aufgabenmodule abbilden lassen. Nach Abel und Bollig [1] ist der modellbasierten Entwicklung ein geeigneter Formalismus zu Grunde zu legen, der klar zwischen Spezifikation und Abarbeitung trennt und einen sukzessiven Entwicklungsprozess unterstützt. Um die Durchgängigkeit bis zur operativen Betriebsphase zu gewährleisten, muss der Formalismus eine Echtzeitsynchronisation sowie externe Prozessanbindungen unterstützen. Weiterhin sollte die Spezifikation nicht nur hersteller- und programmiersprachenunabhängig sein, sondern auch als Steuerungsdokumentation dienen und demgemäß weitestgehend durch grafische Beschreibungsmittel (Kecher [51], Kim et al. [52], Song und Kim [111]) abbildbar sein.

Nach Abel und Bollig [1] besteht im Kontext flexibler Fertigungssysteme (Koren et al. [54] und Tang et al. [115]) zunehmend die Anforderung, dass sich Roboterteams situativ an sich ändernde Prozess- oder Umweltbedingungen anpassen. Diese Anforderung bedingt eine hohe Adaptivität seitens der Robotersteuerung. Aus Sicht der M&S-Theorie stellen hoch flexible Robotersteuerungen sogenannte *Large-Scale-Systems* (Zeigler et al. [131]) dar. Es soll untersucht werden, wie eine Integration der zuvor betrachteten Methoden mit Ansätzen zur Untersuchung von Large-Scale-Systems erfolgen kann und welche Vorbeziehungsweise Nachteile sich daraus ergeben.

1.2 Background

In diesem Abschnitt werden in sechs Unterabschnitten wesentliche Grundlagen eingeführt, auf die in dieser Arbeit aufgebaut wird. Von besonderer Bedeutung im Kontext der Arbeit sind die Analyse von Interaktionen in MRS, die aufgabenorientierte Spezifikation von Robotersteuerungen auf Basis des DEVS Formalismus und die systematische Entwicklung von Robotersteuerungen nach dem SBC Ansatz.

1.2.1 Funktionsweise von Gelenkarmrobotern

Nach Weber [124] und Rokossa [91] sind Gelenkarmroboter Roboter, deren Kinematik sich aus mehreren gelenkig miteinander verbundenen Achsen ergibt und die einen Endeffektor

in Form eines zumeist wechselbaren Werkzeugs führen. Die Achsen und der Endeffektor besitzen getrennt ansteuerbare Antriebe mit interner Sensorik. Weiterhin kann ein Gelenkarmroboter externe Sensorik besitzen. Die Steuerung des Roboters erfolgt durch eine Programmsteuerung, die Bewegungen und Aktionen vorgibt und überwacht. Das Konglomerat aus Hard- und Software eines Gelenkarmroboters wird als Single-Robotersystem (SRS) bezeichnet. Ein Verbund von mehreren SRS, ein sogenanntes Roboterteam, wird in dieser Arbeit analog zu Bielawny et al. [10] und Czarnecki [25] als Multi-Robotersystem (MRS) bezeichnet.

Robotersteuerungen lassen sich anhand ihrer Steuerungsarchitektur in verschiedene Klassen einteilen. Nach Mataric [70] bestimmt die Steuerungsarchitektur die prinzipielle Art und Weise, wie die Steuerung eines Roboters organisiert ist und damit die wesentliche Funktionalität. Demgegenüber teilt Burth [3] Steuerungen nach dem Aktivitätsniveau ein. Hierunter werden gemäß Burth verschiedene Arten des Handelns gegenüber der Umwelt beziehungsweise die Änderung des Verhaltens auf sich ändernde Umweltzustände verstanden. Nach Lunze [61] besitzen Robotersteuerungen eine hierarchische Struktur. Diese folgt der allgemeinen Hierarchie von Supervisory-Control-Systemen nach Groover [42]. Die Supervisory-Control-Ebene entspricht nach Groover der System- oder Zellebene in der Automatisierungstechnik. Die Programmsteuerung eines SRS, die Bewegungen und Aktionen vorgibt und überwacht, bildet die Systemebene einer Robotersteuerung.

1.2.2 Roboterorientierte Middleware

Die Schnittstelle zur direkten Prozesssteuerung wird durch eine roboterorientierte-Middleware (RMW) realisiert. Eine RMW bildet die spezifischen Funktionalitäten eines Roboters durch eine Menge von Befehlen und Datenstruktrukturen ab. Eine universelle, herstellerunabhängige RMW wird seitens der Hersteller nicht unterstützt und langjährige Normierungsbemühungen, wie zum Beispiel die Industrial Robot Language (IRL) [31], werden bis heute durch die Hersteller größtenteils ignoriert. Diese Diversität erschwert die Realisierung von MRS mit SRS verschiedener Hersteller. Um das Problem zu lösen, wurde im Jahr 2007 die Entwicklung des Robot-Operating-System (ROS) [76] durch das Stanford Artificial Intelligence Laboratory initiiert. Seit April 2012 wird ROS unter Führung der gemeinnützigen Organisation der *Open Source Robotics Foundation* [126] weiterentwickelt. Ursprünglich war die Entwicklung von ROS auf den Sektor der mobilen Robotik fokussiert. Mit der Einführung von ROS-Industrial im Jahr 2013 gewinnt ROS auch bei Gelenkarmrobotern an Bedeutung, wenn auch noch nicht im breiten industriellen Umfeld. Im Bereich der Forschung wurden bereits vor der Einführung von ROS-Industrial analoge Ansätze für herstellerübergreifende RMW entwickelt, wie beispielsweise in Christern et al. [20] und Chinello et al. [16] vorgestellt. Hier werden exemplarisch Server für KUKA-Robotercontroller und MATLAB-Clients implementiert. Diese ermöglichen die Steuerung von KUKA-Robotern aus der MATLAB-Umgebung. Eine Weiterführung dieses Konzepts wird in Deatcu [21] gezeigt. Hier wurde die Toolbox aus Maletzki [43] weiterentwickelt und um die Unterstützung von KAWASAKI Robotern erweitert. Zum gegenwärtigen Entwicklungsstand können KUKA und KAWASAKI Roboter, mittels eines abstrahierten einheitlichen Befehlssatzes, vom MATLAB-Interpreter aus programmiert werden. Eine Übersicht weiterer Ansätze zur Steuerungsentwicklung wird in Senfelt [105] gegeben.

1.2.3 Online vs. Offline-Programmierung

Für die Roboterprogrammierung existieren verschiedene Programmiermethoden [124, 67], welche sich grundsätzlich in Online- und Offline-Methoden einteilen lassen. Zur Klasse der Online-Methoden gehören alle Ansätze, bei deren Nutzung das reale Robotersystem benötigt wird. Zur zweiten Klasse zählen alle Ansätze, bei denen das reale Robotersystem nicht während der Steuerungsentwicklung verwendet wird. Das reale Robotersystem wird während der Entwicklungszeit zumeist durch einen virtuellen Stellvertreter ersetzt und die Steuerungsalgorithmen an diesem getestet. Die entsprechenden Softwaresysteme werden als Computer-Aided-Robotics (CAR)-Systeme bezeichnet. Jeder größere Roboterhersteller bietet basierend auf seiner proprietären Entwicklungsumgebung ein spezifisches CAR-System an. Daneben existieren einige herstellerunabhängige CAR-Systeme, wie beispielsweise die Softwaresysteme Visual Components [121] oder EasyRob [34]. Der Schwerpunkt von CAR-Systemen liegt bei Gelenkarmrobotern auf der Kinematiksimulation und deren Visualisierung. Die integrierte Programmiersprache ist roboterorientiert. Bezogen auf den gesamten Entwicklungsprozess einer Robotersteuerung unterstützen CAR-Systeme insbesondere die Inbetriebnahme einer Steuerung.

1.2.4 Systematische Entwicklung und Aufgabenorientierung

Für die systematische Entwicklung komplexer Software- und Automatisierungslösungen wurden spezifische Vorgehensmodelle entwickelt. Das V-Modell wurde ursprünglich für die systematische Entwicklung von IT-Projekten entwickelt und durchlief bis heute mehrfache Standardisierungen [2]. Vorgeschlagen wurde der Ansatz zuerst von dem US-amerikanischen Softwareingenieur Barry Boehm im Jahre 1979. Das Vorgehen basiert auf dem Wasserfallmodell [86]. Phasenergebnisse werden zu verbindenden Vorgaben der nächsttieferen Projektphase. Der RCP-Ansatz nach Abel [1] baut maßgeblich auf den Ideen des V-Modells auf, unterscheidet sich aber in den Anforderungen an die den Steuerungsentwicklungsprozess begleitenden Softwarewerkzeuge. Hier besteht oft ein Problem beim Übergang zwischen den einzelnen Entwicklungsphasen. Der RCP-Ansatz fordert explizite Schnittstellen zwischen den jeweiligen Entwicklungswerkzeugen oder eine den kompletten Entwicklungsprozess begleitende Softwareumgebung, welche auch als Toolkette bezeichnet wird. Der SBC Ansatz ist eine spezielle Vorgehensmethode und ein Framework zur Umsetzung des RCP. In Maletzki [68] wird der SBC-Ansatz erstmalig im Kontext mit der Entwicklung von Robotersteuerungen betrachtet. Der SBC-Ansatz wurde bereits vor dem RCP-Ansatz entwickelt und ist spezifisch auf das Anwendungsgebiet der Steuerungstechnik ausgerichtet.

Aufgabenorientierte Spezifikationen unterstützen nach Weber und Siciliano [124, 107] einen effizienten Steuerungsentwurf (Task-Oriented-Control, TOC). TOC wurde in verschiedenen Formen zur Steuerungsimplementierung bei Single-Robotersystem (SRS) angewendet [124, 67, 102, 56, 101]. Das Grundprinzip von TOC besteht in der Abbildung komplexer Steuerungsprobleme durch eine Menge von Aufgaben, sogenannte Tasks, und deren Verknüpfung. Der Entwurf einer TOC kann nach dem Top-Down (Dekomposition) oder Bottom-Up Prinzip (Komposition) erfolgen. Hierbei gilt das Prinzip des *Closure Under Coupling*. Dies bedeutet nach Zeigler [131], dass eine aus Teilaufgaben komponierte Aufgabe sich nicht von einer gleichwertigen atomaren Aufgabe unterscheiden lässt. In der Arbeit wird die Umsetzung von TOC in Verbindung mit dem des SBC-Ansatz aufgezeigt.

1.2.5 Interaktionen in der Robotik

Die bei SRS erfolgreich eingesetzten Entwicklungsmethoden auf ein Multi-Robotersystem (MRS) zu übertragen stellt eine Herausforderung dar. Dies liegt unter anderem an dem Auftreten von gegenseitigen Wechselwirkungen, sogenannten Interaktionen, zwischen den einzelnen Robotern. Denn eine wesentliche Zielstellung von MRS ist, dass die Roboter Teams bilden, um vorgegebene Aufgaben effizient zu lösen.

Interaktionen treten in der Robotik in drei Bereichen auf. Der erste Bereich stellt die Interaktion von Menschen und Robotern dar, welche gemeinsam an Aufgaben arbeiten. Hierbei unterstützt der Roboter den Menschen und übernimmt zum Beispiel repetierende Aufgaben. Robotersysteme welche für diese Anwendung konzipiert werden, verfügen zumeist über umfangreiche Sensorik, beispielsweise zur Erfassung des Menschen und werden in diesen Zusammenhang als *Cooperative Robots* oder abgekürzt als *Cobots* bezeichnet. Corrales et al. [22] geben einen Überblick über aktuelle Herausforderungen der Steuerungsentwicklung für Cobots, insbesondere in Bezug auf die Sicherheit des Menschen in einem industriellen Umfeld. In Bilberg et al. [11] werden Cobots im Zusammenhang mit *digitalen Zwillingen* betrachtet. Die spezifischen Anforderungen, die Modellierung, Simulation und Anwendung digitaler Zwillinge wird zum Beispiel in [123, 4, 96, 66, 11, 72] diskutiert.

Eine weitere Art von Interaktion tritt bei autonom agierenden mobilen Robotersystemen auf. Diese Robotersysteme bestehen zumeist aus Schwärmen gleicher oder ähnlicher mobiler Basiseinheiten, welche als Team Aufgaben lösen. Die an das Team gestellten Aufgaben sind in der Regel nicht durch einen einzelnen Roboter lösbar, sodass mehrere Einheiten miteinander interagieren müssen, um die Aufgaben zu erfüllen. Wie beispielsweise in Behnke [8] diskutiert wird, besteht die Herausforderung unter anderem in der Kommunikation, Aufgabenzuordnung, Pfadplanung sowie der Adaption des Roboterteams an neue Prozessbedingungen. Makris et al. [65] diskutieren Lösungsansätze auf Basis einer lokalen oder globalen Steuerungsebene, insbesondere mit dem Schwerpunkt der oft hohen Funktionsredundanz der Teammitglieder. In Paker [79] wird ein mathematisches Modell zur Beschreibung des Such- und Lösungsraums, mit dem Schwerpunkt der Bewegungskoordination und Ressourcenteilung einzelner Teammitglieder definiert. In Yan et al. [127] erfolgt eine umfassende Literaturanalyse zum aktuellen Stand der Technik mobiler MRS sowie ein historischer Rückblick auf den Begriff MRS seit den 1980er Jahren.

Der dritte Bereich von Interaktionen betrifft Interaktionen zwischen industriellen Gelenkarmrobotern, die ein MRS bilden. Die einzelnen Roboter sind stationär aufgestellt, auf Portalen verfahrbar oder als Mehrarmrobotersystem konzipiert. Die einzelnen Roboter(arme) besitzen in der Regel teilweise überlappende Arbeitsräume. In Bartelt et al. [7] wird beispielsweise ein Bauteilhandling mit zwei stationären Knickarmrobotern diskutiert. Die Roboterarme sind an einem gemeinsamen Controller angeschlossen und müssen aufgrund der Interoperabilität (Anschlüsse, Programmierung) vom selben Hersteller sein. Die Nicht-Unterstützung von Interoperabilität mit Robotern anderer Hersteller behindert oft die effiziente Realisierung von MRS mit Gelenkarmrobotern.

Eine Klassifikation von MRS wird in Mataric [69] am Beispiel von mobilen Robotern gezeigt. Die Interaktionskonzepte *Koordination* und *Kooperation* der mobilen MRS lassen sich analog auf stationäre Gelenkarmroboter übertragen. Bei anderen Aspekten gibt es teilweise grundlegende Unterschiede. Im Focus der mobilen MRS liegt zuerst die eigene Lokalisierung. Dann folgt die Verarbeitung der eigenen Wahrnehmung und die Fusion

mit den Daten (Position, Wahrnehmungen) der anderen Roboter, um eine gemeinsame Pfadplanung vorzunehmen. Hierfür verfügen mobile Robotersysteme häufig über eine große Anzahl unterschiedlicher Sensoren, mit denen sie ihre Umwelt überwachen. Im Gegensatz zu mobilen MRS verfügen Industrieroboter in der Regel nicht über so umfangreiche Sensorik und werden ortsfest oder auf einem Portal verfahrbar eingesetzt. Damit ist ihr Arbeitsbereich festgelegt und ihre Position auf Basis einer definierten Kinematik berechenbar. Unter der Voraussetzung entsprechender Kommunikationsschnittstellen können die Kinematiken mehrerer Roboter verbunden werden, um Bewegungen zu synchronisieren. In diesem Fall wird von einer geometrischen Kopplung gesprochen (Freymann [38]) oder teilweise von Master/Slave-Systemen (Kosuge und Ishikawa [55]).

Lüth [64] führt an, dass selbständig handelnde Systeme, welche kooperativ und koordiniert zusammenwirken, als Agenten bezeichnet werden. In diesem Zusammenhang stellen Roboter technische Agenten dar, welche nicht nur informationstechnisch, sondern auch physikalisch miteinander wechselwirken. Für zielgerichtete Wechselwirkungen müssen sich die technischen Agenten zeitlich und örtlich abstimmen. Wie Tolk [116] diskutiert, sind autonome Robotersysteme und intelligente Softwareagenten topologisch und konzeptionell verwandt. Dies bedeutet, dass Erkenntnisse beider Anwendungsgebiete aufeinander übertragen werden können. Nach Ritter [112] beschäftigt sich die bisherige Forschung vorwiegend mit den internen Abläufen eines Agenten. Hierunter versteht Ritter die Definition von Zielen, die Repräsentation von Wissen und die logische Entscheidungsfindung. Er merkt an, dass für bestimmte Anwendungsfälle eine Kooperation mehrerer Agenten notwendig ist. Bei der Entwicklung von Multi-Agentensystemen (MAS) ist neben der Kommunikation oft die Koordination der Agenten von Bedeutung.

1.2.6 Formale Methoden und Model-based Design

Der vorteilhafte Einsatz von Formalismen zur Systemspezifikation und der Simulation zur Verifikation von Steuerungsproblemen wurde im Übersichtsaufsatz von Litz und Frey [58] herausgearbeitet. Weiterhin wird auf die mögliche Weiternutzung von Modellen, zum Beispiel durch automatische Codegenerierung, verwiesen. Eine breit akzeptierte Definition des Begriffs *Simulation* erfolgte durch Shannon [106]: *Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system.* Relativ analog dazu definiert die VDI-Richtlinie 3633-1 [32]: *Simulation ist ein Verfahren zur Nachbildung - das bedeutet Modellbildung - eines realen oder gedachten Systems mit seinen internen dynamischen Prozessen in Form eines experimentierbaren Modells, um zu Erkenntnissen zu gelangen, die auf die Realität übertragbar sind. Im weitesten Sinne versteht man unter Simulation auch das Vorbereiten, Durchführen und die Auswertung von Simulationsexperimenten mit einem Simulationsmodell.*

Die von Zeigler 1976 eingeführte DEVS ist ein system-theoretisch-basierter Formalismus zur Spezifikation und Simulation ereignisdiskreter Systeme [132]. Der Formalismus wurde in den vergangenen Jahrzehnten kontinuierlich weiterentwickelt. Als Meilensteinpublikationen gelten die zweite und dritte Auflage des Buches *Theorie of Modeling and Simulation* [131, 135], welche aufbauend auf dem DEVS-Fundament neue Erweiterungen einführen. Der Parallel-DEVS (PDEVS) Formalismus nach Chow [19] ist eine Weiter-

entwicklung des ursprünglichen Classic-DEVS-Formalismus. PDEVS führt neue Mechanismen zur Verarbeitung zeitgleicher Ereignisse ein. Als grafische Repräsentation der Dynamik eines Atomic-DEVS-Modells wird in Song [110] das DEVS-Diagramm eingeführt. Zum Zweck der Prozessanbindung werden in dieser Arbeit fünf DEVS-Erweiterungen auf ihre Eignung im Sinne einer durchgängigen Steuerungsentwicklung untersucht. Der Real-Time-DEVS (RT-DEVS) Formalismus nach Zeigler et al. [131] ist eine Erweiterung des Classic-DEVS-Formalismus. In Sarjoughian und Gholami [95] wird der Action-Level-Real-Time-DEVS (ALRT-DEVS) Ansatz vorgestellt, der grundlegend auf den Ideen des RT-DEVS-Formalismus basiert, aber auch Erweiterungen des Parallel-DEVS-Formalismus nutzt. In Moallemi [73] wird der Real-Time-Embedded-PDEVS (RTE-PDEVS) Formalismus für die schrittweise simulationsgestützte Entwicklung von Hardwareanwendungen eingeführt. Der DEVS-Based-Transparent-Modeling-and-Simulation-Framework (TR-DEVS) Formalismus nach Risco Martin [90] ist ein Ansatz, um ein Classic-DEVS um eine Prozessanbindung zu erweitern. Der PDEVS-RCP-Formalismus ist eine Erweiterung des PDEVS-Formalismus und wird in Schwatinski [103] detailliert beschrieben.

Ausgehend von den Anforderungen der *Industrie 4.0* gilt es, Konzepte zur Realisierung adaptiver und flexibler Steuerungen zu entwickeln, wobei die Terme adaptiv und flexibel in dieser Arbeit synonym verwendet werden. Die Charakteristik einer flexiblen Steuerung besteht in Anlehnung an Weller [125] in der situativen Anpassung der Steuerung an die aktuellen Prozessgegebenheiten. Zur Realisierung flexibler Steuerungen für MRS wird in dieser Arbeit in Analogie zu Maletzki [67] auf das SES/MB Framework aufgebaut. Eine SES ermöglicht die Spezifikation von Wissen in Form eines Graphen, konkret durch einen gerichteten azyklischen Baum. Ein SES-Baum besteht aus Knoten und Kanten, welche Attribute definieren können sowie Links zu Komponenten in einer Modelbasis. Die Grundlagen der klassischen SES-Theorie wurden von Zeigler [133] eingeführt und von Zeigler und Hammonds [134] wesentlich erweitert. Darüberhinaus wird im Rahmen der Realisierung flexibler Steuerungen für MRS auf SES-Erweiterungen nach Pawletta et al. [83] und Folkerts et al. [36] sowie auf das erweiterte SES/MB-Frameworks nach Schmidt [97] Bezug genommen und der SES/MB-Ansatz mit dem SBC-Ansatz fusioniert. In diesem Zusammenhang wird weiterhin auf das von Zeigler [132] originär eingeführte Konzept des Experimental-Frame (EF) zurückgegriffen, welches in Nachfolgearbeiten von Rozenblit [92], Daum und Sargent [26], Traoré und Muzy [118] sowie Schmidt [97] weiterentwickelt wurde. Ganz allgemein bildet ein EF den Kontext eines Modells ab.

1.3 Zielsetzung, Forschungshypothesen und Struktur der Arbeit

Die Analyse des State of the Art zeigte, dass es keinen herstellerunabhängigen Ansatz auf Basis von formalen und modellbasierten Beschreibungsmethoden zur Entwicklung von Robotersteuerungen für MRS gibt. Weiterhin zeigte sich die Bedeutung der Aufgabenorientierung bei der Beschreibung von Robotersteuerungen und die Rolle von Interaktionen in MRS. Die Abbildung von Interaktionen in Form von wiederverwendbaren Aufgaben ist bisher ungelöst. Der DEVS Formalismus hat sich als herstellerunabhängiger und modellbasierter Ansatz bei der Steuerungsspezifikation von SRS in verschiedenen Arbeiten bewährt. Allerdings gibt es Defizite bezüglich der Durchgängigkeit über die einzelnen Phasen der

Steuerungsentwicklung.

Das Ziel dieser Arbeit besteht in der Entwicklung einer DEVS-basierten Methode zur durchgängigen, modellbasierten und herstellerunabhängigen Steuerungsentwicklung für Gelenkarmroboter in MRS. Es wird auf dem SBC-Ansatz aufgebaut, welcher in Maletzki [67] bereits zur durchgängigen Steuerungsentwicklung für SRS eingesetzt wurde. Für MRS müssen Interaktionen zwischen den Robotern bei der Steuerungsentwicklung zusätzlich mitbetrachtet werden. Dazu wird untersucht, wie Interaktionen in Form wiederverwendbarer Aufgaben spezifiziert werden können. Hierfür ist eine Identifikation möglicher Interaktionsklassen notwendig. Zur Erhöhung der dokumentarischen Aussagekraft der formalen DEVS-Spezifikation wird eine neue graphische Notation angestrebt. Nach Einführung und Entwicklung der methodischen Grundlagen sollen ausgewählte Interaktionsklassen anhand von Beispielen untersucht werden.

Ausgehend von der Zielsetzung und der Vorgehensweise können drei Forschungshypothesen formuliert werden, die zu untersuchen sind. Im Verlauf der Bearbeitung entstand eine vierte Hypothese die ebenfalls in der Arbeit untersucht wurde.

Hypothese 1: *Interaktionen zwischen Gelenkarmrobotern in einem MRS können mittels wiederverwendbarer und komponierbarer Aufgaben beschrieben werden.*

Hypothese 2: *Mit dem DEVS-Formalismus kann für MRS eine durchgängige modellbasierte Steuerungsentwicklung erfolgen.*

Hypothese 3: *Der SBC-Ansatz als Vorgehensmodell sowie Software-Framework unterstützt eine systematische Steuerungsentwicklung für MRS und in Verbindung mit einer entsprechenden Middleware eine herstellerunabhängige Entwicklung.*

Hypothese 4: *Auf Basis des SES/MB-Frameworks können strukturiert flexible (adaptive) Steuerungen für MRS realisiert werden.*

Nachfolgend wird mit kurzen kapitelbezogenen Zusammenfassungen ein Überblick über die Struktur und den Inhalt der Arbeit gegeben.

Kapitel 2 beginnt mit der Betrachtung allgemeiner Steuerungsparadigmen für Robotersysteme und den Grundlagen der Roboterprogrammierung bei Verwendung einer RMW. Weiterhin erfolgt eine Klassifikation der Methoden zur Roboterprogrammierung, ausgehend von einer Einteilung in On- und Offline-Methoden. Anschließend werden verschiedene Vorgehensmodelle und Frameworks für einen systematischen Entwicklungsprozess diskutiert. Als besonders relevant für eine durchgängige Steuerungsentwicklung wird der SBC-Ansatz identifiziert, welcher im Zusammenhang mit einer aufgabenorientierten Steuerungsrealisierung näher betrachtet wird. Abschließend wird der Begriff *Interaktion* im Kontext von MRS eingeführt und es werden mögliche *Interaktionsklassen* identifiziert.

Kapitel 3 betrachtet wesentliche Aspekte der ereignisdiskreten Modellierung und Simulation im Zusammenhang mit ereignisdiskreten Steuerungen. Neben der Forderung nach Durchgängigkeit steht die Anforderung der Wiederverwendbarkeit beziehungsweise Erweiterungsfähigkeit von Modellen – möglichst von der frühen Entwurfsphase bis in die Betriebsphase von Steuerungen – im Vordergrund. Hinsichtlich der zielgerichteten und

systematischen Umsetzung automatisierungstechnischer Problemstellungen wird der DEVS-Formalismus untersucht. Zunächst werden grundlegende DEVS-Formalismen vorgestellt. Zur graphischen Repräsentation der Modelldynamik wird in die DEVS-Diagramm-Notation eingeführt und diese anschließend um neue Beschreibungsmittel erweitert. Abschließend werden weitere DEVS-Ausprägungen zur Echtzeit- und Prozessanbindung bezüglich der gestellten Anforderungen untersucht.

Kapitel 4 zeigt einen neuen Ansatz zur durchgängigen Entwicklung von Steuerungen für MRS. Dieser basiert auf dem SBC-Ansatz und den zuvor vorgestellten DEVS Formalismen. Es wird ein neuer modifizierter DEVS-Formalismus, genannt PDEVS-RCP-V2, entwickelt. Anhand eines Fallbeispiels wird eine durchgängige Steuerungsentwicklung mit dem neuen Ansatz aufgezeigt. Im Anschluss erfolgt die Betrachtung von MRS und insbesondere von Interaktionen zwischen den Robotersystemen. Es wird gezielt die Abbildung von Interaktionen auf Aufgaben im Sinne einer aufgabenorientierten Steuerung untersucht. Für die zuvor eingeführten Interaktionsklassen werden konzeptionelle Lösungsansätze entwickelt.

Kapitel 5 untersucht einen neuen Ansatz zur Steuerungsspezifikation und Steuerungsgenerierung für flexible (adaptive) MRS-Steuerungen auf Basis des SES/MB-Frameworks. Zuerst werden Grundlagen des SES/MB-Frameworks sowie spezifische Erweiterungen eingeführt und anschließend wird konzeptionell die Anwendung im Kontext dieser Arbeit betrachtet.

Kapitel 6 diskutiert die softwaretechnische Umsetzung von TOC Applikationen für MRS auf Basis der untersuchten Methoden. Zuerst wird auf die Verwendung von MATLAB als SBC-Plattform sowie die benutzten und zum Teil erweiterten Toolboxen eingegangen. Anschließend werden prototypische Implementierungen der eingeführten Interaktionsklassen anhand von Fallbeispielen aufgezeigt. Es erfolgt eine detaillierte Strukturierung der Steuerungen gemäß dem SBC-Ansatz. Die Spezifikation der einzelnen Komponenten erfolgt auf Basis des PDEVS-RCP-V2-Formalismus unter Verwendung der erweiterten DEVS-Diagramm-Notation. Abgeschlossen wird das Kapitel mit einer Gegenüberstellung von Lösungsvarianten, auf Basis der klassischen Vorgehensweise beziehungsweise unter Verwendung des SES/MB-Frameworks, für ein komplexes Fallbeispiel einer flexiblen Steuerung.

Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf mögliche weiterführende Arbeiten.

2 Entwurf, Programmierung und Inbetriebnahme von Robotersteuerungen

Mit der Industrie 4.0 gewinnen heterogene Robotersysteme auch für kleine und mittelständische Unternehmen (KMU) an Bedeutung. Die Robotersysteme müssen für vielfältige Aufgaben programmiert werden. Dies bedingt einen systematischen Entwicklungsprozess, beginnend mit einem Steuerungsentwurf auf Basis einer festzulegenden *Steuerungsarchitektur*. Die Realisierung des Steuerungsentwurfs erfolgt anschließend durch die Programmierung der Steuerungssoftware.

Die Programmierung einer Robotersteuerung kann auf unterschiedliche Arten erfolgen, bedingt aber immer den Einsatz einer *roboterorientierten-Middleware*, welche die Funktionalitäten des Robotersystems durch eine Menge von Computerbefehlen abbildet. Eine Programmierung kann mit und ohne Verwendung des physischen Roboters erfolgen. Hierbei wird von *On- und Offline-Methoden zur Roboterprogrammierung* gesprochen. Offline-Methoden ermöglichen die Programmierung ohne physischen Zugriff auf den Roboter und unterstützen den Einsatz von computergestützten *Vorgehensmodellen zur Steuerungsentwicklung*. Diese können den Steuerungsentwickler bei der Erstellung, Bewertung und möglichst frühzeitigen Erprobung der Steuerungsalgorithmen helfen.

Zunächst werden mögliche Steuerungsarchitekturen sowie roboterorientierte-Middleware vorgestellt. Im darauf folgenden Unterabschnitt werden die Methoden der Roboterprogrammierung erläutert. Im Weiteren erfolgt eine Einführung in computergestützte Vorgehensmodelle und Frameworks. Danach wird die Umsetzung des *aufgabenorientierten Steuerungsentwurfs*, am Beispiel des *Simulation Based Control Ansatzes* gezeigt. Abschließend wird der Begriff *Interaktion* im Kontext von *Multi-Robotersystemen* eingeführt und mögliche *Interaktionsklassen* identifiziert.

2.1 Steuerungsarchitektur

Ein Roboter ist ein autonomes System, bestehend aus Hard- und Softwarekomponenten, dessen Grad an Autonomie sich aus dem Zusammenspiel von Sensorik, Aktorik und Steuerung ergibt. Bei Gelenkarmrobotern besteht die Hardware mindestens aus einem Roboterarm mit Werkzeug und einem Controller. Der Controller verarbeitet die durch Sensorik erfassten Messdaten und steuert die zum Roboter gehörende Aktorik. Der Controller und sein Steuerungsprogramm sind in einem vernetzten Automatisierungssystem Elemente einer zumeist hierarchisch aufgebauten Prozesssteuerung. Die Steuerungsentwicklung folgt in der Regel einem Steuerungsparadigma. Dieses beeinflusst das mögliche

Aktivitätsniveau und ist entscheidend für die Leistungsfähigkeit der Steuerung. Die Begriffe Steuerungsparadigma und Aktivitätsniveau werden nachfolgend eingehender diskutiert.

2.1.1 Steuerungsparadigma und Aktivitätsniveau

Wie aus sensorischen Messdaten eine Ansteuerung der Aktorik erfolgt, wird durch die zugrundeliegende Steuerungsarchitektur festgelegt (Weber [124], Lunze [60], Mataric [70], Groover [42]). Die Steuerungsarchitektur bestimmt die prinzipielle Art und Weise wie die Steuerung eines Roboters organisiert ist. Zur Auswahl einer Steuerungsarchitektur sollte der Steuerungsentwickler alle Randbedingungen, unter welchen das Robotersystem zu funktionieren hat, erfassen und analysieren. Von besonderer Relevanz für die Auswahl der Steuerungsarchitektur sind die verfügbaren Ressourcen (Arbeitsraum, Zeit, Daten, usw.) und problemspezifische Anforderungen. Dieser Zusammenhang ist in Abbildung 2.1 schematisch dargestellt.

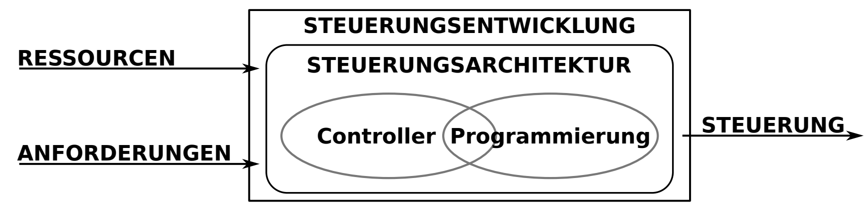
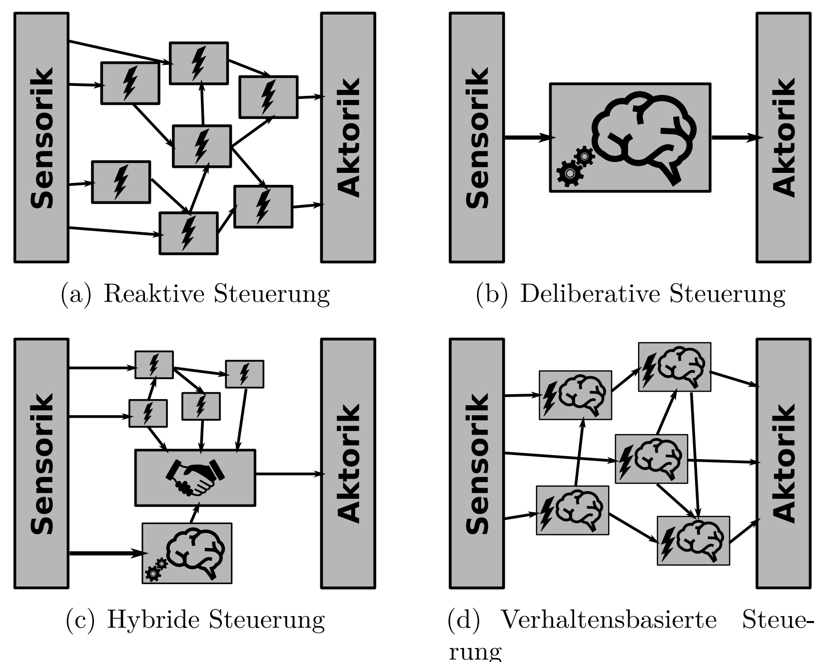


Abbildung 2.1: Einflussgrößen und Zielgröße der Steuerungsentwicklung

Nach Weber [124] sind die Komponenten Sensorik und Aktorik Bestandteil einer jeden Steuerungsarchitektur (vgl. Abb. 2.2). Sensorik steht dabei für alle Komponenten, die für das Erfassen und die Vorverarbeitung der Sensordaten zuständig sind. Die Komponente Aktorik steuert die Aktoren des Roboters.

Nach Mataric sind vier Steuerungsarchitekturen, wie in Abbildung 2.2 gezeigt, zu unterscheiden. Mit jeder Steuerungsarchitektur kann ein bestimmtes Steuerungsparadigma realisiert werden. Nachfolgend werden die Steuerungsparadigmen vorgestellt. Weiterhin erfolgt eine prinzipielle Bewertung, ob das jeweilige Steuerungsparadigma Lernfähigkeit unterstützt. Lernfähigkeit ist nach Weller [125] wie folgt definiert:

Lernfähigkeit ist die autonome Erlangung einer Schrittfolge, auch Steuerfluss genannt, zur Lösung eines Problems. Der Lernprozess kann unter Verwendung einer belehrenden Instanz (dem Lehrer) oder ohne a priori Wissen, durch eine selbstständige Bewertung der Reaktion der Umwelt auf eine Aktion erfolgen.



(a) Reaktive Steuerung

(b) Deliberative Steuerung

(c) Hybride Steuerung

(d) Verhaltensbasierte Steuerung

Abbildung 2.2: Mögliche Varianten einer Steuerungsarchitektur und zugehöriges Steuerungsparadigma schematisch entworfen auf Basis von Mataric [70]

Teilbild 2.2 (a): **Reaktive Steuerungen** sind besonders für eine sich ständig ändernde Umwelt geeignet. Hierbei werden Sensoreingangswerte direkt an Aktorwerte gekoppelt. Die Steuerungsarchitektur basiert auf einer Menge vernetzter Blöcke, welche auf Eingangsgrößen durch die sofortige Berechnung von Ausgangsgrößen reagieren. Eine Ausgangsgröße eines Blocks kann hierbei zu einer Eingangsgröße eines anderen Blocks werden. Der Ansatz kann auch als "Denke nicht nach, reagiere einfach" beschrieben werden. Ein Limit dieses Ansatzes ist, das für jeden möglichen Sensorwert eine dazugehörige Aktion hinterlegt werden muss. Reaktive Steuerungen können nicht lernen, da sie kein Gedächtnis besitzen. Ein wiederkehrender, "sensorischer Reiz" führt bei erneutem Auftreten immer zur selben Reaktion der Steuerung.

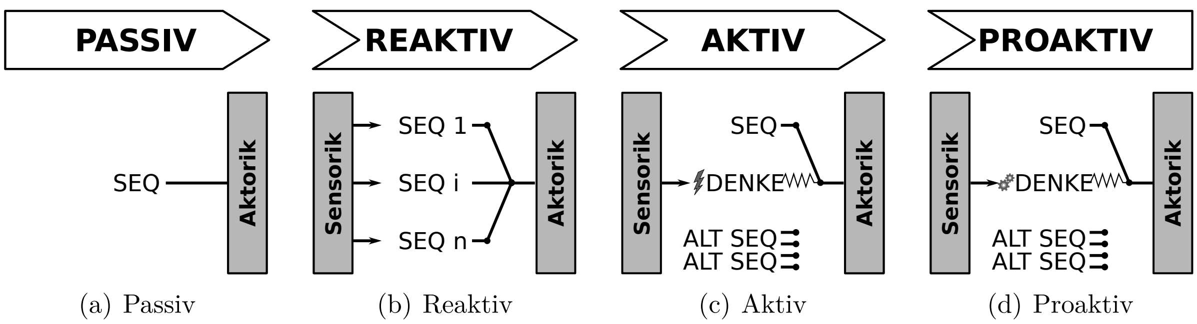
Teilbild 2.2 (b): **Deliberative Steuerungen** verwenden alle zur Verfügung stehenden Sensorwerte und zuvor erlangtes Wissen über die Umwelt, um eine bestmögliche Abfolge von Aktionen zu planen. Um dieses Ziel zu erreichen, muss die Steuerung alle hinterlegten Pläne analysieren, um den Plan zu finden, welcher eine gegebene Zielstellung bestmöglich erfüllt. Hierfür muss die Steuerung die Konsequenz einer zukünftigen Aktion abschätzen können. In der Regel benötigt dieser Prozess lange Rechenzeit und macht sofortige Reaktionen auf sich ändernde Umweltbedingungen unmöglich. Die Devise ist hierbei: "Denke hart und dann handle". Unter der Bedingung, dass die Umwelt für einen längeren Zeitraum vorhersehbare oder konstante Eigenschaften aufweist, kann der deliberative Steuerungsansatz zum Lernen einer bestmöglichen Schrittfolge von Aktionen genutzt werden.

Teilbild 2.2 (c): **Hybride Steuerungen** wurden entwickelt, da deliberative Steuerungen viel Rechenzeit für die Erstellung einer geeigneten Abktionsfolge benötigen. Bei diesem Ansatz werden die zwei zuvor beschriebenen Ansätze miteinander kombiniert. Eine Komponente der Steuerung ist hierbei reaktiv und die andere vorausplanend. Die Herausforderung liegt darin, beide Komponenten miteinander zu verknüpfen und dabei auftretende Konflikte

zu lösen. Hierfür wird zumeist eine dritte vermittelnde Komponente benötigt. Die Devise ist: "reagiere und plane parallel zueinander". Wie zuvor bei der deliberativen Steuerung, kann eine bestmögliche Steuerungsfolge erlernt werden. Jedoch ermöglicht der reaktive Anteil der Steuerung eine sofortige Reaktion auf sich plötzlich ändernde Umweltänderungen. Dies ermöglicht eine Unterbrechung und Anpassung der zuvor verfolgten Steuerungsabfolge, falls dies situationsbedingt notwendig wird.

Teilbild 2.2 (d): **Verhaltensbasierte Steuerungen** sind von der Biologie inspiriert und versuchen, das Verhalten von Tiergehirnen beim Umgang mit harten Problemen des gleichzeitigen Denkens und Handelns zu imitieren. Wie auch beim hybriden Steuerungsansatz ist die Steuerung modularisiert. Jedoch unterscheiden sich die Module nicht grundlegend. Jedes Modul bildet ein autonomes Verhalten ab, kann Eingaben erhalten und Ausgaben berechnen, welche zu Eingaben anderer Module werden können. Durch die Vernetzung der Module und den Austausch von Informationen kann die Steuerung Konsens erreichen und vorausschauend Aktionen planen. Im Gegensatz zum hybriden Ansatz wird kein zentraler Planer verwendet. Die Komplexität der Module kann sich stark unterscheiden, sodass schnelle Reaktionen oder auch komplexe Verhaltensweisen realisierbar sind. Weiterhin kann ein Modul Informationen über die Umwelt akkumulieren und basierend auf dem erlangten Wissen, neue Verhaltensmuster lernen. Die Leistungsfähigkeit dieses Ansatzes wird in der Literatur [70] als gleichwertig zum hybriden Ansatz beschrieben. Die Devise ist: "Denke in die Richtung in der du handelst".

Ein weiterer Ansatz zur Steuerungsklassifikation wird in Burth [3] gezeigt und basiert auf dem *Aktivitätsniveau* einer Steuerung. Hierunter werden gemäß Burth verschiedene Arten des Handelns gegenüber der Umwelt beziehungsweise die Änderung des Verhaltens auf sich ändernde Umweltzustände verstanden. Die Umwelt ist im betrachteten Kontext die Prozessumgebung des Robotersystems und wird durch Sensorik überwacht. Burth [3] teilt Steuerungen nach dem verwendeten Aktivitätsniveaus in vier Kategorien ein, welche in Abbildung 2.3 schematisch dargestellt sind. Die Anpassungsfähigkeit an sich ändernde Umweltweltbedingungen nimmt von Teilbild (a) zu Teilbild (d) zu.



(a) Passiv (b) Reaktiv (c) Aktiv (d) Proaktiv

Abbildung 2.3: Varianten von Steuerungsarchitekturen in Anlehnung an Burth [3]

Teilbild 2.3 (a): **Passive Steuerungen** verhalten sich gegenüber Änderungen der Umgebung passiv. Die Steuerung ergreift keinerlei Initiative bezüglich einer Anpassung der Befehlssequenz (SEQ) an sich ändernde Umweltbedingungen. Beim Auftreten von Störungen werden keinerlei Gegenmaßnahmen zur Beseitigung der Ursachen ergriffen. Ein typisches Beispiel ist eine einfache Ablaufsteuerung, welche einem festen zuvor programmierten Algorithmus folgt. Folglich verfügen *Passive Steuerungen* in der Regel über keine

Sensoren zur Überwachung ihrer Prozessumgebung.

Teilbild 2.3 (b): **Reaktive Steuerungen** verändern ihren Ablauf, wenn sie durch einen externen Reiz dazu angeregt werden. Externe Reize sind durch Sensorik erfasste Prozesszustände oder -ereignisse, welche zur Ausführung einer anderen Befehlssequenz (SEQ) führen. Die Menge der möglichen alternativen SEQ ist zur Laufzeit der Steuerung unveränderlich und wurde durch den Steuerungsentwickler im Vorfeld festgelegt. Die Steuerung kann die Ursache einer Störung nicht aktiv beseitigen. Die Definition des Verhaltens einer *Reaktiven Steuerung* ist nach [3] und [70] identisch.

Teilbild 2.3 (c): **Aktive Steuerungen** reagieren nicht nur auf eine externe Störung, sondern versuchen die Ursache für das Problem aktiv zu beseitigen. In einem ersten Schritt wird zunächst auf den anliegenden Reiz reagiert und in weiteren Schritten die Reizursache beseitigt. Hierfür benötigt die Steuerung Umweltwissen, da eine geeignete Aktion (ALT SEQ) zur Beseitigung der Reizursache ausgewählt werden muss. Nach Mataric [70] gibt es drei Ansätze zur Umsetzung einer aktiven Steuerung. Für den Fall, dass der Steuerung ausreichend Zeit zur Planung einer neuen Befehlsabfolge zur Verfügung steht, kann eine *deliberative Steuerung* verwendet werden. Muss die Steuerung jedoch sehr schnell auf einen Reiz mit einer Aktion antworten, so sollten *hybride* oder *verhaltensbasierte Steuerungen* verwendet werden.

Teilbild 2.3 (d): **Proaktive Steuerungen:** versuchen durch eine möglichst frühzeitige, differenzierte Planung und zielorientiertes Handeln, denkbare Prozesszustände vorauszusehen und diese vorab in einer gewünschten Form zu beeinflussen. In Einzelfällen können die Grenzen zwischen aktiven und proaktiven Steuerungen fließend ineinander übergehen. Zur Umsetzung dieses Aktivitätsniveaus sind *hybride* oder *verhaltensbasierte* Steuerungen geeignet. Die Vorhersage kann mittels modellbasierten Ansätzen erfolgen.

Abbildung 2.4 zeigt schematisch den Zusammenhang zwischen den Steuerungsparadigmen nach Mataric [70] und den Aktivitätsniveaus nach Burth [3]. Hierbei fällt auf, dass der Anwendungsfall *passive Steuerung* in Mataric nicht betrachtet wurde.

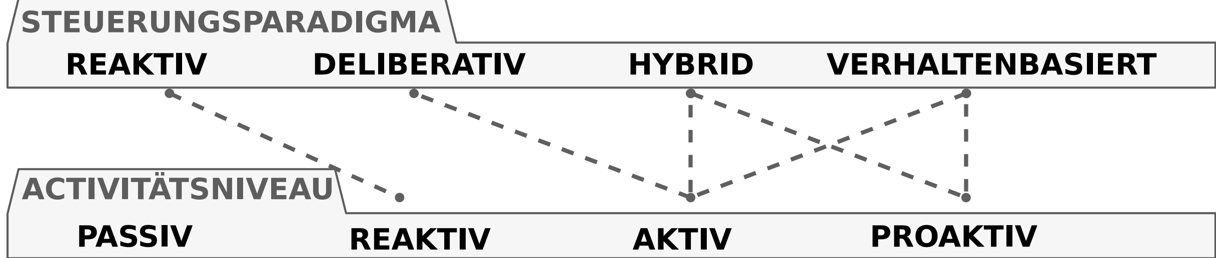


Abbildung 2.4: Zusammenhang zwischen Steuerungsparadigmen nach Mataric [70] und Aktivitätsniveaus nach Burth [3]

Zwischenfazit: Jeder der vorgestellten Ansätze findet in der heutigen Steuerungsentwicklung Anwendung und hat seine jeweiligen Vor- und Nachteile. Dabei unterscheiden sich die vorgestellten Ansätze in ihrer Komplexität und Leistungsfähigkeit stark von einander. Während passive Steuerungen sich durch reine Aktionssequenzen realisieren lassen, besitzen proaktive Steuerungen mit integrierten Vorhersagemodellen eine komplexe Steuerungsarchitektur. Generell steigt mit zunehmendem Aktivitätsniveau die Komplexität und Entwicklungszeit der Steuerung. *Denken ist langsam*, *reagieren muss schnell sein*,

Denken ermöglicht es, Aktionen vorausschauend zu planen und damit gefährliche Situationen zu vermeiden. Jedoch erfordert *Denken* eine Vielzahl detaillierter Informationen über die Prozessumgebung. Diese stehen nicht immer zur Verfügung und können sich bei dynamischen Prozessen während des *Denkens* ändern.

2.1.2 Roboterorientierte-Middleware

Roboter sind mechatronische Systeme und verfügen als solche über Sensorik und Aktorik. Um ein Robotersystem für eine Aufgabe einzurichten muss eine Steuerung realisiert werden. Diese verarbeitet durch einen Algorithmus (Steuerungslogik) die Eingangssignale und erzeugt hieraus Ausgangssignale zur Ansteuerung der Aktorik und externen Schnittstellen.

Nach Lunze [61] besitzen Robotersteuerungen eine hierachische Struktur, wie in Abbildung 2.5 gezeigt. Diese folgt der allgemeinen Hierarchie von Supervisory-Control-Systemen nach Groover [42]. Die Supervisory-Control-Ebene entspricht nach [42] der System- oder Zellebene in der Automatisierungstechnik. In der Robotik spezifiziert die Handlungsplanung die Steuerungslogik des Roboters auf Systemebene. Die Schnittstelle zur direkten Prozesssteuerung wird durch eine RMW realisiert.

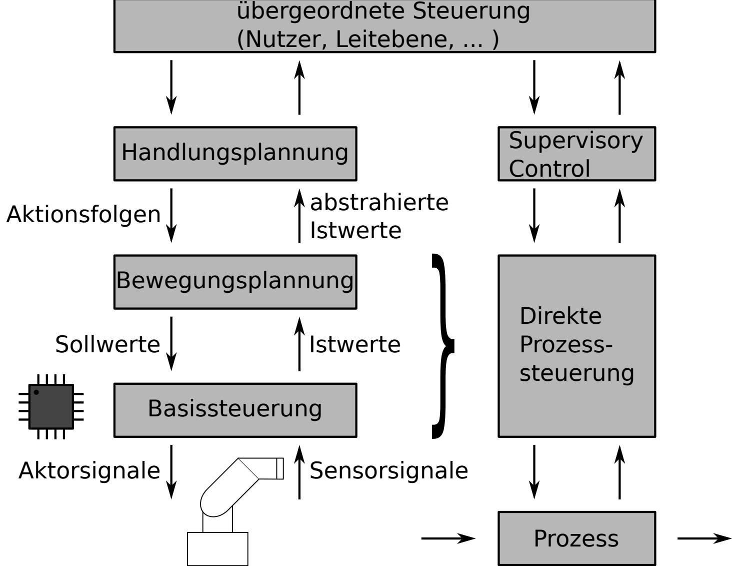


Abbildung 2.5: Linke Seite: Hierarchische Struktur von Robotersteuerungen nach Lunze [61]
Rechte Seite: Allgemeine Struktur einer Supervisory-Control nach Groover [42]

Die RMW bildet die spezifischen Funktionalitäten des Roboters in Form von Computerbefehlen und Datenstrukturen ab. Sie ermöglicht beispielsweise das programmgesteuerte Auslesen von Sensordaten oder das Bewegen eines Roboterarms. Wie im vorherigen Unterabschnitt gezeigt wurde, enthalten alle nicht passiven Steuerungen die Komponenten Sensorik und Aktorik. Diese sind elementare Bestandteile einer Steuerung und dienen zum Erfassen sensorischer Messwerte und zum Steuern der Aktorik. Unabhängig von der zu realisierenden Steuerungsarchitektur bildet die RMW die Grundlage zur Entwicklung der Komponenten Sensorik und Aktorik nach Abbildung 2.2.

Neben unterschiedlichen Robotertypen, welche hardwaretechnisch für bestimmte Anwendungsgebiete optimiert sind, unterscheiden sich auch die Softwarelösungen zum Pro-

grammieren von Robotern stark und sind oft herstellerspezifisch. Die Hersteller schaffen sich dabei zunehmend ihr eigenes in sich geschlossenes Ökosystem, welches ausschließlich die Roboter, Sensoren und Aktoren der eigenen Marke unterstützt. Die Programmierung der Roboter erfolgt zumeist mittels einer herstellerspezifischen Programmiersprache, wie in Abbildung 2.6 schematisch für drei Roboterhersteller dargestellt. Hieraus folgt, dass die RMW von Hersteller A in der Regel mit Hersteller B inkompatibel ist.

Herstellerspezifische Programmiersprachen können für den Entwicklungsingenieur ein Problem darstellen, denn obwohl sich der Funktionsumfang der Industrieroboter[1] zwischen den Herstellern kaum unterscheidet, muss dieser sich in unterschiedliche Softwareumgebungen einarbeiten. Weiterhin können einmal entwickelte Steuerungslösungen nicht einfach auf verschiedene Robotersysteme portiert werden und die Steuerungsentwicklung unterliegt den zugrundeliegenden Restriktionen der jeweiligen herstellerspezifischen Softwareumgebung.

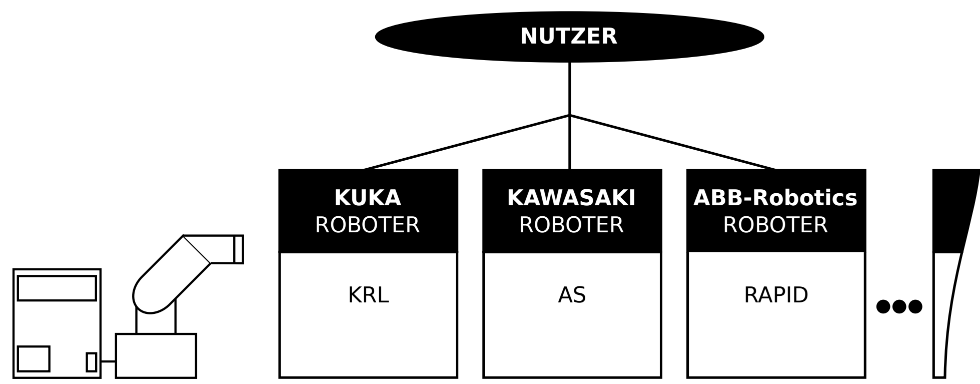


Abbildung 2.6: Ausgewählte Roboterhersteller und ihre Programmiersprachen

Langjährige Normierungsbemühungen wie die IRL [31] werden durch die Roboterhersteller bis heute größtenteils ignoriert. Diese Gegebenheit führt zu Einschränkungen in der gemeinsamen Nutzung von Robotern unterschiedlicher Hersteller beziehungsweise macht diese unmöglich. Insbesondere Applikationen, bei denen mehrere Roboter interagieren müssen, sind in der Regel auf die Hard- und Softwarelösung eines Herstellers begrenzt.

Seit dem Jahr 2007 befindet sich das ROS [76] in der Entwicklung, welches durch das Stanford Artificial Intelligence Laboratory initiiert wurde. Seit April 2012 wird ROS von der gemeinnützigen Organisation Open Source Robotics Foundation [126] weiterentwickelt. Auch in der klassischen Industrierobotik gewinnt ROS zunehmend an Bedeutung. Seit 2013 beschäftigt sich das ROS Industrial Consortium mit der Unterstützung von ROS für Industrieroboter. Jedoch befindet sich ROS nach wie vor in der Entwicklung. Die Software-Schnittstellen unterliegen noch entwicklungsbedingten Änderungen. Im Gegensatz zu Sprachnormierungen wie der IRL, definiert ROS ein Framework mit Schnittstellen, welches die Integration proprietärer Software-Lösungen unterstützt.

Ein anderer Ansatz zur herstellerunabhängigen Programmierung von Robotern basiert auf dem Client-Server Modell. Bei diesem Ansatz agiert der Robotercontroller als Server und gibt einem Client Zugriff auf sämtliche herstellerspezifischen Dienste des Roboters. Client und Server kommunizieren über die Hardwareschnittstellen (Seriell, Ethernet, usw.) des Robotercontrollers. Steuerbefehle des Clients werden in einem ersten Schritt in ein

[1]Industrieroboter steht in der Arbeit Synonym für Gelenkarmroboter

Datenwort übersetzt und anschließend über die Datenverbindung an einen Interpreter übertragen. Der Interpreter wird in der herstellerspezifischen Sprache des Roboterherstellers entwickelt und auf dem Robotercontroller ausgeführt. Der Interpreter bringt das das empfangene Datenwort auf dem Robotercontroller zur Ausführung und kann Datenworte an den Client zurücksenden. Auf diese Weise können Sensordaten kommuniziert werden.

In Christern et al. [20] und Chinello et al. [16] wurden exemplarisch Server für KUKA-Robotercontroller und MATLAB-Clients implementiert. Im Rahmen dieser Arbeit wurde die Toolbox aus Maletzki [43] weiterentwickelt und um die Unterstütztung von KAWASAKI-Robotern erweitert. Zum gegenwärtigen Entwicklungsstand [21] können KUKA und KAWASAKI Roboter, mittels eines einheitlichen Befehlssatzes, vom MATLAB-Interpreter aus programmiert werden. Der abstrahierte Befehlssatz unterstützt nahezu alle Funktionalitäten der beiden herstellerspezifischen Sprachen. Die prinzipelle Softwarearchitektur ist in Abbildung 2.7 dargestellt. Sie gestattet eine einfache Erweiterbarkeit auf andere Robotertypen. Eine Übersicht weiterer Ansätze zur Steuerungsentwicklung wird in Senfelt [105] gezeigt.

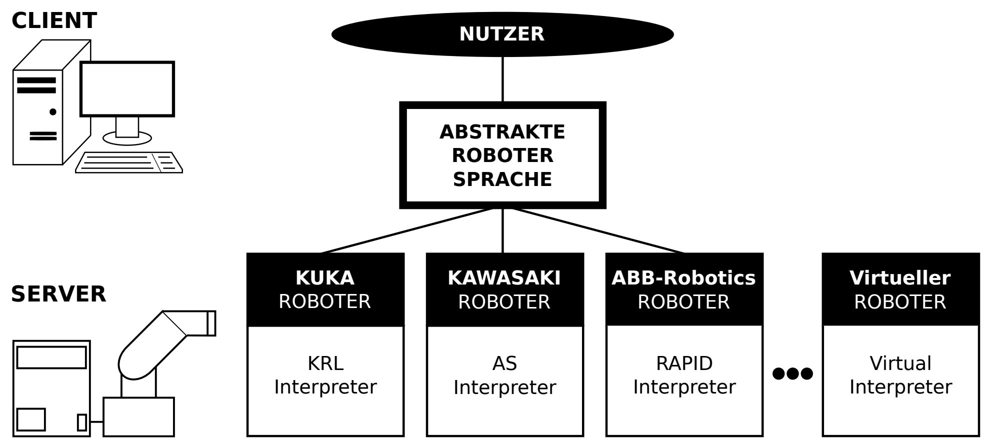


Abbildung 2.7: Steuerungsentwicklung nach dem Client-Server Modell

Nachfolgend werden verschiedene Methoden zur Programmierung von Robotersteuerungen analysiert und nach einen eigenen Schema geordnet. Die Methoden basieren auf einer roboterorientierten-Middleware, welche aufgrund von Abstraktionen zum Teil verdeckt sind. Dennoch werden werden durch die Middleware wesentliche Aspekte, wie zum Beispiel die Portierbarkeit, festgelegt.

2.2 Methoden der Roboterprogrammierung

Für die Roboterprogrammierung existieren verschiedene Programmiermethoden [124, 67]. Abbildung 2.8 zeigt eine Einordnung der Methoden in die Klasse Online-Methoden beziehungsweise Offline-Methoden. Zur Klasse der Online-Methoden gehören alle Ansätze bei deren Nutzung das reale Robotersystem benötigt wird. Zur zweiten Klasse zählen alle Ansätze bei denen das reale Robotersystem nicht während der Steuerungsentwicklung verwendet wird. Daneben existieren hybride Methoden, welche eine Kombination von Online- und Offline-Methoden darstellen und im Kontext dieser Arbeit nicht weiter betrachtet werden.

2.2.1 Online-Methoden

Hierunter fallen alle Methoden bei welchen die Programmierung direkt mit dem Robotersystem erfolgt. Der Roboter steht für die Zeit der Steuerungsentwicklung nicht anderweitig zur Verfügung. Da die Entwicklung direkt mit dem Roboter erfolgt, müssen spezifische Sicherheitsaspekte beachtet werden. Allgemein liegt der Vorteil von Online-Methoden in der besonderen Anschaulichkeit der Programmierung.

Programmierung durch Beispiele

Hierunter werden alle Methoden verstanden, bei denen der Roboter an gewünschte Positionen verfahren wird und diese dann zur Wiederholung gespeichert werden.

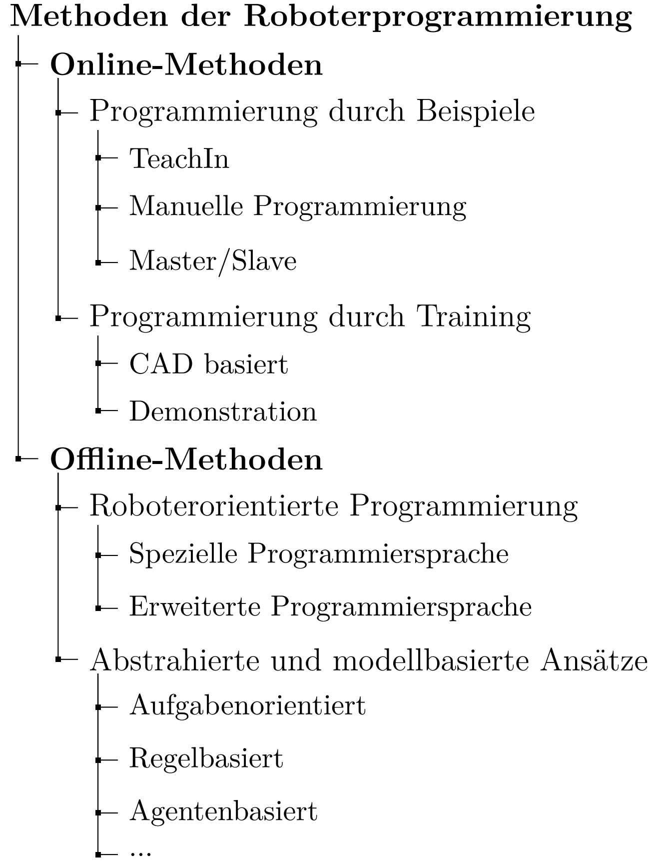


Abbildung 2.8: Typische Methoden der Roboterprogrammierung

TeachIn: Die Ansteuerung des Roboters erfolgt durch ein Bediengerät. Es können entweder interaktiv Kommandos abgesetzt und ausgeführt werden oder der Roboter wird mithilfe einer 3D-Maus in die gewünschte Position verfahren.

Manuelle Programmierung: Hierunter versteht man die direkte Führung des Roboters durch den Bediener am Endeffektor. Häufig wird diese Art der Programmierung durch einen Kraft-Momentensensor realisiert. Das Robotersystem speichert automatisch die durch den Bediener vorgegebene Position ab, sodass eine anschließende Wiederholung einer Bewegungsfolge möglich ist.

Master/Slave: Bei diesem Verfahren handelt es sich um eine manuelle Programmierung eines Master-Roboters als Stellvertreter. Dieser wird entlang einer beliebigen Bewegungsbahn

geführt, während zeitgleich automatisch Zwischenpunkte gespeichert werden. Anschließend erfolgt eine automatische Wiederholung durch den Slave-Roboter.

Programmierung durch Training

Hierunter werden alle Ansätze zusammengefasst, bei denen die Bewegungsbahnen und Aktionen zur Beginn der Programmierung nur mit einer sehr geringen Genauigkeit vorgegeben werden. Durch das anschließende Ausführen und sensorische Erfassen der Abweichung vom Sollverhalten ist die Robotersteuerung in der Lage, iterativ das gewünschte Maß an Genauigkeit zu erreichen. Die Optimierung der Bahnplanung kann beispielsweise durch Lernalgorithmen erfolgen.

CAD basiert: Das Steuerungsprogramm zur Umsetzung der Bewegungsbahn wird direkt aus einer Konstruktionszeichnung (CAD Datei) abgeleitet. Anschließend erfolgt die Optimierung der erzeugten Bewegungsplan am Robotersystem. Die Verfeinerung der zunächst groben Bewegungsfolge bedarf eines geeigneten Messsystems. Dieses liefert erforderliche Daten, um die aktuelle Bewegung an die Sollbewegung des Roboters anzupassen. Unter Umständen wird ein manuelles Eingreifen des Menschen in den Bewegungsablauf erforderlich.

Demonstration: Mittels *Vormachen* durch den Menschen (Demonstration) werden sensorisch Bewegungsbahnen erfasst. Eine Möglichkeit stellt die Beobachtung des Menschen durch ein Kamerasystem dar. Hierbei kopiert das Robotersystem wesentliche Bewegungsabläufe des Menschen, um diese in einen nachgelagerten Verarbeitungsschritt genauer zu detaillieren. Die Verfahrensklasse Demonstration ist ein aktueller Forschungsgegenstand. Auf dem Markt befinden sich aber bereits erste Robotersysteme, die durch die Beobachtung eines Menschen, selbstständig eine geeignete Bewegungsfolge ableiten können.

2.2.2 Offline-Methoden

Bei Offline-Methoden wird der Roboter während der Steuerungsentwicklung nicht benötigt, dass heißt die Programmentwicklung erfolgt offline an einem vom Robotersystem unabhängigen Computer. Während der Entwicklung kann das Robotersystem weiterhin für bestehende Applikationen verwendet werden, sodass keine Stillstandszeiten anfallen. Exakter Weise muss an dieser Stelle darauf hingewiesen werden, dass auch bei Offline-Methoden in der Phase der Inbetriebnahme einer Steuerung der Roboter benötigt wird.

Roboterorientierte Programmierung

Die roboterorientierte Programmierung setzt direkt auf die Funktionalitäten der roboterorientierten Middleware, gemäß Unterabschnitt 2.1.2, auf. Dem Steuerungsentwickler stehen eine Menge von Befehlen in Form von einer Programmiersprache zur Verfügung. Typische Funktionalitäten eines Roboters, wie das Verfahren an eine neue Position (Bewegungsbefehle), das Auslesen von Sensoren (Datenakquise) oder das Ansteuern externer Hardware, werden durch entsprechende Befehle abgebildet. Weiterhin stehen Daten- und Programmablaufstrukturen analog zu höheren Programmiersprachen zur Verfügung. Weiterhin können Konzepte zur Hierarchisierung und Kapselung von Programmcode, Bestandteil einer roboterorientierten Programmiersprache sein.

Bei der roboterorientierten Programmierung erfolgt die Steuerungsentwicklung durch das Schreiben eines Programms durch den Entwickler. Die Steuerungsentwicklung kann

hierbei mit einer allgemeinen, erweiterten Programmiersprache oder durch eine spezielle zumeist herstellerspezifische Programmiersprache erfolgen.

Spezielle Programmiersprache: Die Sprachen sind häufig an das Roboterökosystem des Herstellers angepasst. Steuerungen für herstellerfremde Robotersysteme können damit nicht entwickelt werden. Synonym kann auch von herstellerspezifischen Programmiersprachen gesprochen werden.

Erweiterte Programmiersprache: Die Roboterprogrammierung erfolgt mittels einer allgemeinen Programmiersprache. Diese ist zumeist in Form von Softwarebibliotheken um roboterorientierte Sprachelemente erweitert.

Automatisierungsprojekte können komplexe Problemstellungen beinhalten, die eine Steuerungsentwicklung durch Entwicklerteams bedürfen. Fehlende, aber notwendige Absprachen zwischen den Teammitgliedern führten in der Vergangenheit häufig zu Fehlern. Ein Ziel der abstrahierten und modellbasierten Ansätze besteht darin, Konzepte der Softwareengineerings auf die Steuerungsentwicklung zu übertragen. Damit wird versucht die Modularität, Wiederverwendbarkeit und Testbarkeit zu verbessern. Die nachfolgenden Ansätze abstrahieren die zugrundeliegende Problemstellung durch ein ausgewähltes Paradigma.

Abstrahierte und modellbasierte Ansätze

Hierunter werden alle Verfahren verstanden, die auf einer höheren Abstraktion basieren und bei der Problemlösung stärker zwischen Konzept und Umsetzung unterscheiden. Zu den abstrahierten Ansätzen werden nachfolgend drei in der Robotik etablierte Methoden betrachtet. Formal stellen die abstrahierten Methoden oft modellbasierte Ansätze dar. In der Robotik wird der Term modellbasiert allerdings oft im Kontext von CAR Systemen gesehen, auf welche danach eingegangen wird.

Aufgabenorientierte Programmierung: Nach Maletzki [67], Kugelmann [56] und Meissner [71] wird bei diesem Ansatz die Steuerungslogik auf Basis vordefinierter Aufgabenmodule beschrieben. Die konkreten Bewegungsabläufe sind nicht unmittelbarer Bestandteil der Aufgabenmodule und werden erst in einem nachfolgenden Verarbeitungsschritt realisiert. Das Abstraktionsprinzip der Aufgabenorientierung ist weit verbreitet in der Robotik und wird in einen späteren Kapitel ausführlich behandelt.

Regelbasierte Ansätze: Bei diesem Ansatz werden die Automatisierungsaufgabe und ihre Prämissen als logische Aussagen (Regeln) formuliert. Eine Prämisse bezeichnet in diesem Kontext eine Voraussetzung oder Annahme. Sie ist eine Aussage, aus der eine logische Schlussfolgerung gezogen wird. Ein Interpreter versucht dann, eine gewünschte Lösungsaussage mithilfe von Interferenzregeln aus der Menge der Prämissen, herzuleiten. Schlussendlich werden die mit den Regeln der Lösungsaussage verknüpften Anweisungen ausgeführt. Die regelbasierten Ansätze haben ihren Ursprung im Forschungsgebiet der künstlichen Intelligenz. Beispiele sowie weitere Informationen zu regelbasierten Ansätzen in der Robotik werden in Lunze [59], van Harmelen et al. [119] und Fraser et al.[37] vorgestellt.

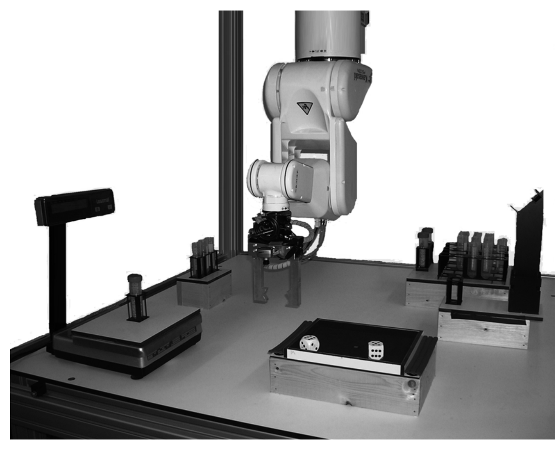
Agentenbasierte Ansätze: Bei agentenbasierten Ansätzen wird die Steuerung durch eine Menge miteinander und mit der Umgebung kommunizierender Software-Agenten realisiert. Ein Software-Agent ist ein Computerprogramm, dass zu eigenständigen, spezifischen Verhalten auf Basis innerer Zustände imstande ist. Jeder Agent versucht ein

Ziel durch das Ausführen von Aktionen zu erreichen. Hierfür kann jeder Agent über eine eigenständige Abbildung seiner momentanen Umgebung verfügen, welche auf Annahmen basiert. Bekommt ein Agent einen Reiz durch seine Umgebung oder andere Agenten, kann dieser mit einer bestimmten Aktion reagieren. Weiterhin kann ein Agent Aktionen unabhängig von externen Reizen ausführen, um sein Ziel zu erreichen. Ein Agent kann sich sowohl reaktiv als auch proaktiv gegenüber der Umwelt verhalten. Ähnlich einem Team, können Agenten miteinander wechselwirken, um ihre Aufgaben zu erfüllen. Insbesondere in den letzten zwei Jahrzehnten wurden im Rahmen der *Künstlichen Intelligenz* vielfältige Agentenansätze entwickelt (Russel und Norvig [93]) und auf Probleme der mobilen Robotik angewendet (Behnke [8]). Anwendungen im Bereich von Industrierobotern sind bisher nicht verbreitet.

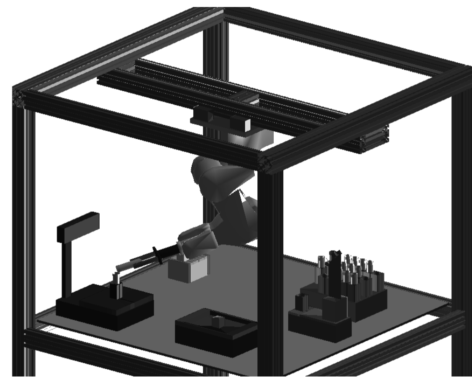
CAR-Systeme CAR-Systeme bieten eine Entwicklungsumgebung zum Schreiben der Steuerungsalgorithmen und eine Prozessvisualisierung. Die Visualisierung bildet alle für den Prozess relevanten Komponenten sowie die zu steuernden Roboter als 3D-Modelle, inklusive ihrer Dynamik, ab. Die Grundlage der 3D-Modelle bilden Computer-Aided-Design (CAD) Dateien, welche durch entsprechende Schnittstellen in das CAR-System eingelesen werden. Der entwickelte Steuerungsalgorithmus kann, vor der Inbetriebnahme am realen Robotersystem, virtuell erprobt und verbessert werden. Weiterhin werden CAR-Systeme zunehmend während der Betriebsphase als digitale Zwillinge eingesetzt.

CAR-Systeme werden von diversen Herstellern angeboten. Jedoch sind insbesondere die CAR-Systeme der meisten Roboterhersteller proprietär und unterstützen nur die Visualisierung der eigenen Robotersysteme. Ein Beispiel hierfür ist die Software KUKA.Sim der Firma KUKA. Ein Steuerungsentwickler ist häufig dazu gezwungen sich auf einen Hersteller und seine Entwicklungswerkzeuge zu spezialisieren. Daneben existieren einige herstellerunabhängige CAR-Systeme, wie beispielsweise die Software *Visual Components* [121] oder *EasyRob* [34]. Hier erfolgt die Steuerungsentwicklung nach wie vor mit der herstellerspezifischen Programmiersprache. Jedoch wird die simultane Programmierung und 3D-Visualisierung unterschiedlicher Robotersysteme diverser Hersteller unterstützt. Ein Problem der nicht von den Roboterherstellern stammenden CAR-Systeme ist, dass Roboterhersteller nicht generell die Dynamik ihrer Robotersysteme offen legen. Dies bedeutet, das Annahmen, wie sich das reale Robotersystem verhält getroffen werden müssen. Die getroffenen Annahmen können zur Problemen bei der späteren Inbetriebnahme der Steuerung führen.

Ein Beispiel für eine 3D-Prozessvisualisierung, welche im Rahmen der Arbeit entwickelt wurde, zeigt Abbildung 2.9. Die Anwendung basiert auf einem durch die Forschungsgruppe CEA entwickelten CAR-System für MATLAB. Dieses ermöglicht die herstellerunabhängige Entwicklung und virtuelle Erprobung der Steuerungsalgorithmen vor der Inbetriebnahme und die Nutzung der modellbasierten Virtualisierung als digitalen Zwilling. Hinsichtlich der Eigenschaften und des Einsatzes von digitalen Zwillingen sei beispielsweise auf [123, 4, 96, 66, 11, 72] verwiesen.



(a) reale Prozessumgebung



(b) virtuelle Prozessumgebung

Abbildung 2.9: Vergleich zwischen realer und virtueller Prozessumgebung

Zwischenfazit: Ohne computergestützte Werkzeuge zur Umsetzung von Automatisierungslösungen müsste eine Steuerungsentwicklung direkt am automatisierungstechnischen Prozess erfolgen. Ein wesentlicher Vorteil von Offline-Methoden ist die Steuerungsentwicklung ohne unmittelbare Nutzung der realen Hardware. Dadurch werden existierende Anlagen nicht in ihren laufenden Betrieb blockiert beziehungsweise die Steuerungsentwicklung kann vor der physikalischen Verfügbarkeit einer Anlage beginnen. Ein weiterer Vorteil ist die Möglichkeit der virtuellen Inbetriebnahme, welche ein gefahrloses Testen von Steuerungen unterstützt. Die Verbindung von Offline-Methoden mit CAR-Systemen ermöglicht das Testen im Kontext des gesamten zu automatisierenden Prozesses. Allerdings erfordert die Anwendung ein höheres ingenieurtechnisches Know-how. Aus diesem Grund ist ein systematisches Vorgehen notwendig. Im Bereich der Automatisierungstechnik haben sich deshalb Vorgehensmodelle etabliert, welche den Anwender während der gesamten Steuerungsentwicklung unterstützen. Hierbei erfolgt die Umsetzung systematisch und in mehreren Schritten. Das Ziel ist den Anwender bei der Realisierung einer qualitativ hochwertigen Steuerung zu unterstützen.

2.3 Vorgehensmodelle und Frameworks

In der Steuerungstechnik sind Vorgehensmodelle computergestützte Methoden zur zielgerichteten Umsetzung einer automatisierungstechnischen Aufgabe. Ein Vorgehensmodell begleitet den Entwickler während der gesamten Steuerungsentwicklung. Ein wesentliches Prinzip ist die Vermeidung von Fehlern durch frühzeitige Tests. Hierfür wird der Entwicklungsprozess in mehrere Phasen eingeteilt. Jede Phase verfolgt ein konkretes Zwischenziel. Weiterhin erfordern einige Ansätze die Einhaltung einer bestimmten Steuerungsstruktur durch ein zugrundeliegendes Framework. Ein Framework teilt Komponenten einer Steuerung in funktionale Bereiche ein. Nachfolgend werden drei zur Entwicklung von Robotersteuerungen benutze Ansätze vorgestellt.

2.3.1 V-Modell

Das V-Modell wurde ursprünglich für die systematische Entwicklung von IT-Projekten entwickelt und durchlief bis heute mehrfache Standardisierungen [2]. Vorgeschlagen wurde der Ansatz zuerst von dem US-amerikanischen Softwareingenieur Barry Boehm im Jahre 1979. Das Vorgehen basiert auf dem Wasserfallmodell [86]. Phasenergebnisse werden zu verbindenden Vorgaben der nächsttieferen Projektphase. Heute findet das V-Modell auch in der Automatisierungstechnik Anwendung. Das V-Model teilt den gesamten Entwurfsprozess in einzelne ineinander übergehende Phasen ein, welche durch ausführliche Tests begleitet werden.

Abbildung 2.10 zeigt eine vereinfachte Darstellung des V-Modells nach Maletzki [67]. Der linke Pfad beschreibt die funktionale und technische Spezifikation der Problemstellung in der Entwurfsphase. Die Spezifikation der Problemstellung bildet die Grundlage für den Entwurf einer ersten Steuerungsstrategie. Diese wird zunächst ausführlich getestet und bewertet. Tritt hierbei ein Fehler zum Vorschein, wird ein Entwicklungsschritt wiederholt. Ergibt die Bewertung der Steuerungsalgorithmen, dass diese zur Lösung der Problemstellung geeignet sind, beginnt die Phase der Implementierung. Hierbei werden die entwickelten Steuerungsalgorithmen für die Zielhardware implementiert. Zu diesem Zeitpunkt ist der Detaillierungsgrad am höchsten, da der Steuerungsentwickler jeden Algorithmus individuell für jede zu steuernde Hardwarekomponente anpassen muss. Die nachfolgende Phase besteht aus Komponententests. Jede entwickelte Einzelkomponente wird individuell bezüglich der zuvor festgelegten Spezifikation des linken Astes im V-Modell getestet.

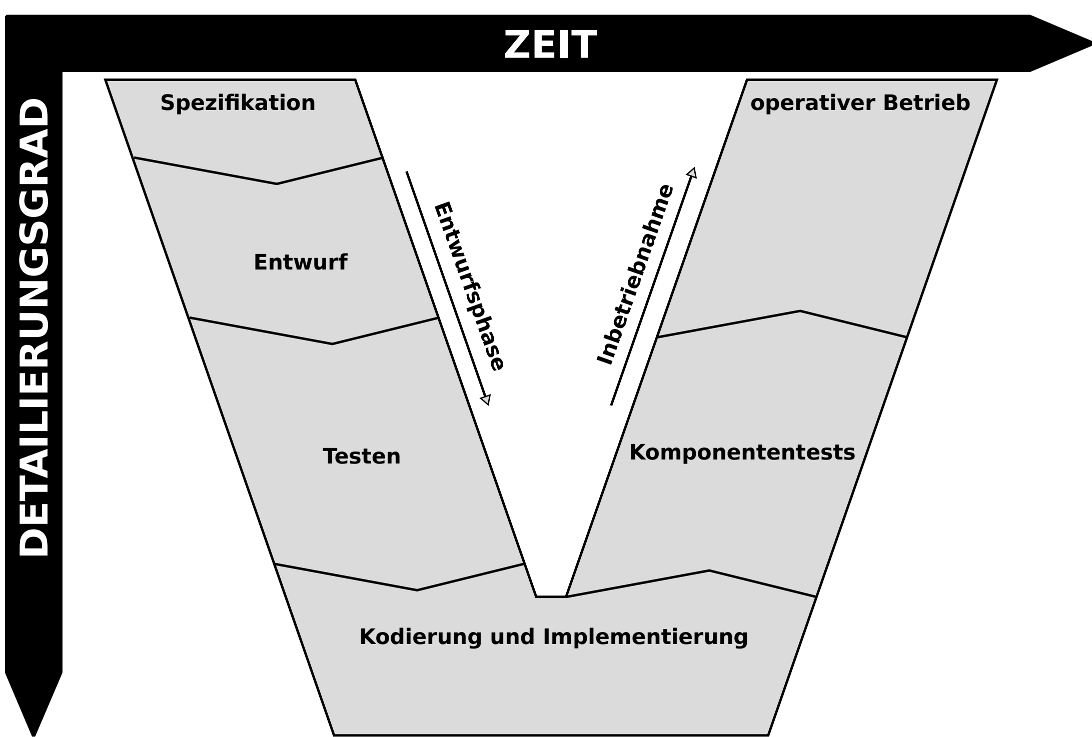


Abbildung 2.10: Realisierung einer Automatisierungslösung nach dem V-Modell gemäß Maletzki [67]

Durch diese Gegenüberstellung soll eine möglichst hohe Testabdeckung erreicht werden, weil die Spezifikationen der jeweiligen Entwicklungsstufen zugleich die Grundlage für

die Testfälle der entsprechenden Teststufe sind. Treten hierbei Probleme zum Vorschein, müssen einzelne Entwicklungsschritte wiederholt werden. Der finale Schritt ist durch den Test aller Komponenten im Wechselspiel miteinander gekennzeichnet. Hierbei erfolgt auch der Übergang in den operativen Betrieb der Steuerung.

In Maletzki [67] wird das V-Modell in Verbindung mit der Offline-Programmierung von Robotersteuerungen präzisiert. Die Entwicklung beginnt wiederum mit der Spezifikation der Steuerungsaufgabe (vgl. Abbildung 2.11). Maletzki empfiehlt neben dem Robotersystem auch die Prozessumgebung des Roboters zu betrachten. Der nächste Schritt ist der Entwurf und die Bewertung einer Steuerungsstrategie durch einen Plannungsingenieur. Dabei wird ein Robotersystem in der Regel als Element eines Materialflusssystems betrachtet. Häufig kommen hierbei diskret-ereignisorientierte Simulationssysteme zum Einsatz, mit denen die Leistungsfähigkeit der zu untersuchenden Steuerungsstrategien bewertet wird.

In der nächsten Phase werden sowohl die Steuerungsalgorithmen als auch der Prozess mit einem Entwurfssystem, heute meist ein CAR-System, modelliert und getestet. Dies ist die Aufgabe des Steuerungsentwicklers. Die Kodierung und Implementierung von ausführbaren Code für die Zielhardware kann nach Maletzki [67] durch eine explizite oder implizite Codegenerierung erfolgen. Bei der expliziten Codegenerierung wird die komplette Robotersteuerung durch Cross-Compiling in ausführbaren Code für die Zielhardware überführt. Die implizite Codegenerierung basiert auf dem im Abschnitt 2.1.2 beschriebenen Client-Server-Ansatz. Abbildung 2.11 zeigt die implizite Codegenerierung mit der horizontalen *Kommunikation* zwischen CAR-System im linken V-Pfad und der Steuerung im rechten V-Pfad. Anschließend erfolgen Komponententests bevor in den operativen Betrieb übergegangen wird.

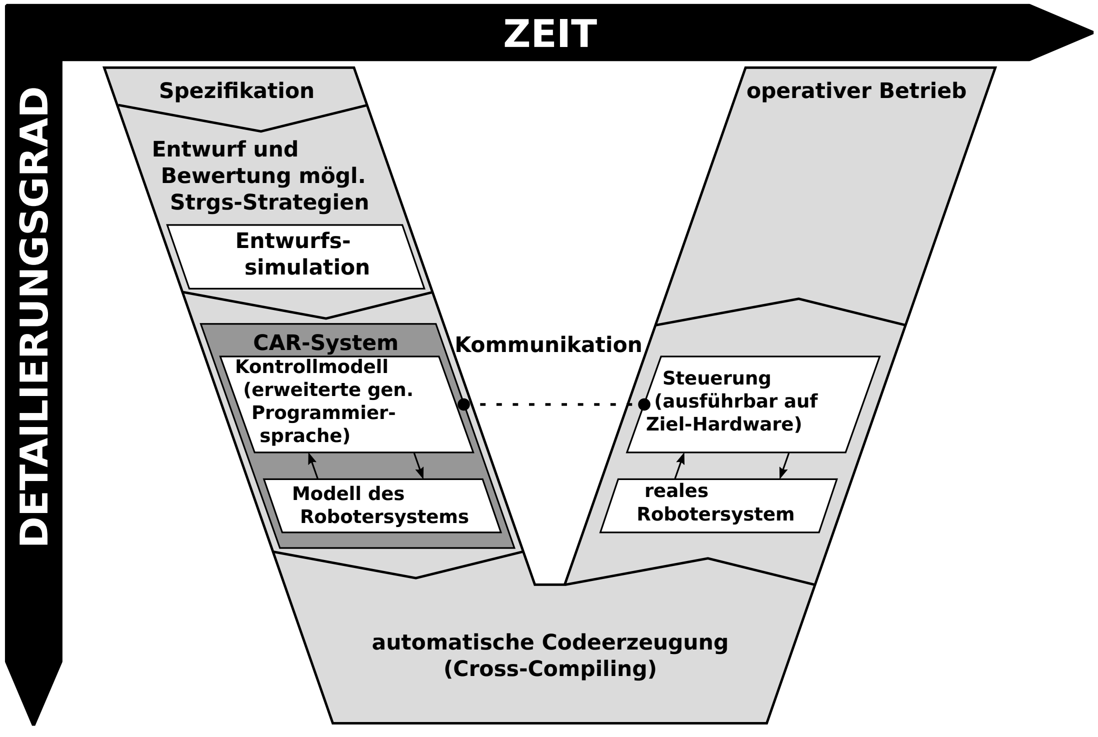


Abbildung 2.11: Angepasstes V-Model zur Robotersteuerungsentwicklung von Maletzki [67]

2.3.2 Rapid-Control-Prototyping (RCP)

Der RCP-Ansatz nach Abel [1] baut maßgeblich auf den Ideen des V-Modells auf, unterscheidet sich aber in den Anforderungen an die den Steuerungsentwicklungsprozess begleitenden Softwarewerkzeuge. Zur Umsetzung der einzelnen Entwicklungsstufen des V-Modells werden unterschiedliche Softwarewerkzeuge eingesetzt. Ein Problem besteht beim Übergang zwischen den einzelnen Entwicklungsphasen. Hier treten erfahrungsgemäß die meisten Fehler auf. Ein Grund für dieses Problem liegt in der händischen Übernahme erzielter Ergebnisse in die nächste Entwicklungsphase beziehungsweise beim Wechsel der phasenbegleitenden Entwicklungssoftware. Der RCP-Ansatz fordert explizite Schnittstellen zwischen den jeweiligen Entwicklungswerkzeugen oder eine den kompletten Entwicklungsprozess begleitende Softwareumgebung, welche auch als Toolkette bezeichnet wird.

Um eine möglichst hohe Testabdeckung zu ermöglichen und damit eine größtmögliche Qualität zu gewährleisten, werden sowohl die Steuerungseinheit (Control Unit) als auch die Prozessumgebung so früh wie möglich durch Simulationsmodelle abgebildet. Eine erste Steuerungsentwicklung kann somit vollständig durch Simulationsmodelle erfolgen, indem Steuerungsalgorithmen entwickelt und mithilfe der simulierten Prozessumgebung erprobt werden. Dieser Vorgang wird als Systemsimulation bezeichnet und ist in Abbildung 2.12 schematisch dargestellt. In einem nächsten Schritt können die identifizierten Steuerungsalgorithmen gegen den realen Prozess getestet werden. Die Steuerung läuft auf dem Entwicklungsrechner, welcher über eine Schnittstelle mit der realen Prozessumgebung kommuniziert und diese steuert. Dieses Prinzip wird nach Abel [1] als Software-in-the-Loop (SiL) bezeichnet. Es gibt weitere Definitionen von SiL, wie zum Beispiel von Ayed et al. [5] oder von Silano et al. [108]. In dieser Arbeit wird auf die Definition von Abel aufgebaut.

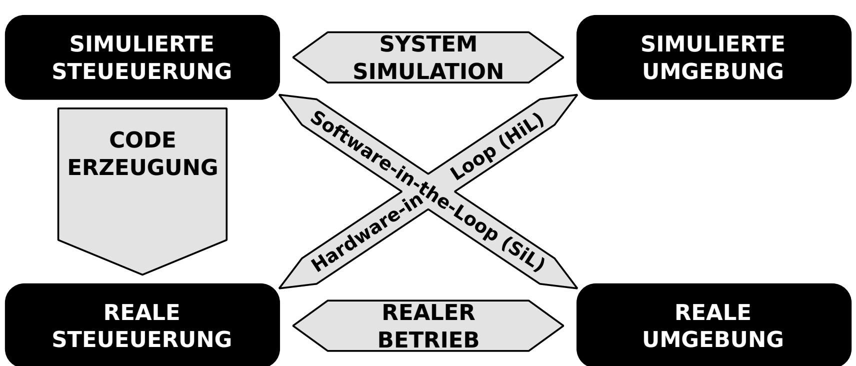


Abbildung 2.12: Verbindung von Systemsimulation, Hard- und Software in the Loop nach Abel [1]

Sind die zugrundeliegenden Steuerungsalgorithmen erfolgreich getestet, so können diese mittels einer automatischen Codegenerierung in spezifischen Steuerungscode überführt werden, welcher anschließend auf einer gewünschten Zielplattform zur Ausführung gebracht wird. Um wiederum eine größtmögliche Testabdeckung zu erreichen, kann die reale Steuerungseinheit zunächst mit der zuvor entwickelten virtuellen Prozessumgebung getestet werden. Nach Abel wird dies als Hardware-in-the-Loop (HiL) Prinzip verstanden. Sind alle der zuvor beschriebenen Testszenarien erfolgreich abgeschlossen, kann die entwickelte Steuerung am realen Prozess in Betrieb genommen werden.

Um auf dem Zielsystem ausführbaren Steuerungscode zu erzeugen, können wie im

vorangegangenen Abschnitt diskutiert, die explizite oder implizite Codegenerierung eingesetzt werden. Gemäß den Anforderungen des RCP-Ansatzes ist eine softwaretechnische Durchgängigkeit zu gewährleisten. Im Fall der expliziten Codegenerierung erfolgt eine Steuerungsentwicklung entweder durch eine integrierte Softwareumgebung oder mit aufeinander abgestimmten Softwarewerkzeugen (Toolkette). Diese müssen zueinander kompatible Schnittstellen zum Austausch von Daten aufweisen. Es muss für die finale Steuerungsimplementierung ein Cross-Compiler existieren, um Steuerungscode für spezifische Hardware zu erzeugen. Bei der impliziten Codegenerierung wird der Entwicklungsrechner im operativen Betrieb zur Ausführung der Steuerungsalgorithmen verwendet. Anstelle einer vom Roboterhersteller abhängigen proprietären Software können offene Entwicklungsumgebungen genutzt werden. Diese ermöglichen unter anderem eine Integration unterschiedlichster Hardwarekomponenten im zu steuernden Prozess. Weiterhin kann die Steuerungsentwicklung weitgehend unabhängig von der Zielplattform erfolgen. Die Kommunikation zwischen dem Entwicklungrechner und der Prozess-Hardware wird mit einer Roboter-Middleware gemäß Abschnitt 2.1.2 realisiert. Eine konkrete Implementierung einer impliziten Codegenerierung unter Verwendung der Entwicklungsumgebung MATLAB wird in Deatcu [21] vorgestellt.

2.3.3 Simulation-Based-Control (SBC) Ansatz

Der SBC-Ansatz ist ein spezielle Vorgehensmethode und ein Framework zur Umsetzung des RCP. In Maletzki [68] wird der SBC-Ansatz erstmalig im Kontext mit der Entwicklung von Robotersteuerungen betrachtet. Weiterhin erwähnt der Autor, dass sich der SBC-Ansatz bereits in unterschiedlichen steuerungstechnischen Anwendungsfeldern bewährt hat. Maletzki verweist auf eine Reihe von Projekten, die sich bereits seit den 1990er Jahren mit der durchgängigen Unterstützung von Entwicklungsprozessen für Steuerungsprobleme befassen. Dazu gehören auch eine Reihe von Projekten, die seit 1994 in einer Kooperation des Instituts für Automatisierungstechnik der Universität Rostock und der Forschungsgruppe Computational Engineering und Automation der Hochschule Wismar durchgeführt wurden [39, 38, 103, 85, 68, 67]. Im Rahmen dieser Arbeiten wurden die Grundlagen zur Entwicklung des SBC-Ansatzes gelegt. Der SBC-Ansatz wurde bereits vor dem RCP-Ansatz entwickelt, ist aber auf das Anwendungsgebiet der Steuerungstechnik ausgerichtet. Analog zum RCP ist das Ziel ein durchgängiger Entwicklungsprozess von der frühen Spezifikation (Entwurf) bis zum operativen Betrieb. Der Entwicklungsprozess erfolgt über alle Entwicklungsphasen hinweg auf einer gemeinsamen Softwareplattform. Eine schematische Darstellung des SBC-Ansatzes ist in Abbildung 2.13 dargestellt.

Der SBC-Ansatz ist eine Entwurfsmethodik sowie ein Framework, welches auf einer spezieller Form des SiL-Prinzips aufbaut. Ein in der Entwurfsphase entwickeltes Simulationsmodell (SM) wird schrittweise bis zu einem operativen Steuerungsprogramm (Control-Software, CS) erweitert und simulativ getestet. Damit entfällt die Reimplementierung von SM in Steuerungscode, wodurch Fehler vermieden werden, Entwicklungszeit eingespart wird und insgesamt die Entwicklungskosten reduziert werden. Um diese Form der durchgängigen Softwareentwicklung zu ermöglichen, bedarf es einer durchgängigen Softwarekette. Der durchgängige Einsatz von SM im Entwicklungsprozess ermöglicht es, Fehler in den Algorithmen frühzeitig zu erkennen und zu beheben. Wie in Abbildung 2.13 dargestellt, ist es notwendig möglichst früh während der Entwicklung konsequent zwischen dem Kontrollmodell (Control-Model, CM) mit der Steuerungslogik und dem Prozessmodell (PM)

mit dem Abbild des realen Prozesses zu unterscheiden. Der operative Betrieb auf Basis des SiL-Prinzips benötigt ein Interface zum realen Prozess, welches gemessene Sensorwerte aufbereitet und umgekehrt Aktorbefehle an die Komponenten des realen Prozesses sendet. Gemäß Abbildung 2.13 sollte für diese Ebene ein Interfacemodell (IM)[102] entwickelt werden. Dieses sollte ein simulatives Testen unterstützen und zugleich auch als Schnittstelle zum realen Prozess agieren. Die Integration des PM in die CS ermöglicht die Berechnung nicht oder schlecht messbarer Prozessgrößen sowie die Realisierung von Beobachterkonzepten. Im Gegensatz zu anderen steuerungstechnischen RCP-Ansätzen ist das PM, als Bestandteil der Steuerung, auch im operativen Betrieb enthalten.

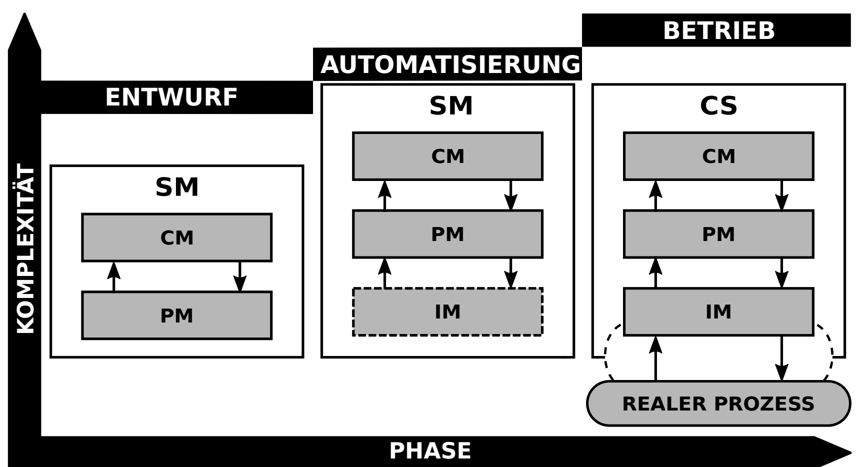


Abbildung 2.13: Schematische Darstellung der SBC-Vorgehensweise für die Steuerungsentwicklung

Nachdem Vorgehensmodelle zur durchgängigen Entwicklung von automatisierungstechnischen Steuerungen vorgestellt wurden, soll anschließend die Übertragung auf eine der vorgestellten Offline-Programmiermethoden gezeigt werden. Es wird die Verbindung des aufgabenorientierten Steuerungsentwurfs (Task-Oriented-Control, TOC) und des SBC-Ansatz gezeigt. Von den abstrahierten Methoden nach Abschnitt 2.2.2 wurde der aufgabenorientierte Ansatz ausgewählt, da er im Bereich der Robotersteuerungen seit mehreren Jahrzehnten eine hohe Relevanz besitzt. Die aufgabenorientierte Abstraktion ist auf viele Problemstellungen der Robotik anwendbar. Weiterhin unterstützt sie die Wiederverwendbarkeit von einmal definierten Aufgaben sowie die Komposition von Aufgaben zur Bildung neuer Ausgaben.

2.4 Aufgabenorientierte Steuerungen (TOC) mit dem SBC-Ansatz

Nachfolgend wird das Konzept der aufgabenorientierten Steuerungen im Detail vorgestellt und die Verbindung zum SBC aufgezeigt. Es werden das Grundprinzip und die Schichten einer TOC sowie ihre Umsetzung mittels SBC-Ansatz dargestellt.

2.4.1 Grundprinzip und Schichten einer TOC

TOC ist ein etabliertes Konzept zum Steuerungsentwurf [124, 107] und wird bereits zur Programmierung von Single-Robotersystemen (SRS) angewendet [124, 67, 102, 56, 101]. Das Grundprinzip besteht in der Abbildung komplexer Steuerungsprobleme durch eine Menge von Aufgaben, sogenannte Tasks, und deren Verknüpfung. Tasks sind zumeist voneinander unabhängige Arbeitsschritte. Einmal identifizierte Tasks werden miteinander verknüpft, um das Steuerungsproblem abzubilden. Die Verknüpfung kann sequentiell, bedingt oder in einer Schleife erfolgen. Durch Verknüpfung können aggregierte Aufgaben gebildet werden. Eine Aufgabe ist jeweils als ein Teilmodell innerhalb des CM umzusetzen. Dies ermöglicht eine Aggregation von einfachen Subaufgaben zu komplexen Aufgaben, indem diese bei Bedarf miteinander verkoppelt werden. Hierbei gilt das Prinzip des *Closure Under Coupling* [131]. *Closure Under Coupling* bedeutet, dass eine durch Teilaufgaben komponierte Aufgabe sich nicht von einer gleichwertigen atomaren Aufgabe unterscheiden lässt. Diese Eigenschaft ist die Basis zur Modularisierung von Aufgaben. Weiterhin kann es zur Problembeschreibung notwendig sein, Tasks mehrfach zu verwenden oder parallel abzuarbeiten. Das Vorgehen beim Erstellen einer TOC entspricht der menschlichen Denkweise beim Lösen komplexer Problemstellungen.

Abbildung 2.14 zeigt den TOC-Ansatz am Beispiel der Problemstellung des Transports von Bauteilen. Das Transportproblem ist in die beiden Tasks PickPart und PlacePart unterteilt. Die Aufgabe PickPart spezifiziert die Aufnahme eines Bauteils an einer definierten Position und die Aufgabe PlacePart beschreibt das Ablegen an einer neuen Position. Die Aufgabe PickPart muss vor der Aufgabe PlacePart ausgeführt werden, was einer seriellen Abarbeitung entspricht. Weiterhin zeigt die Abbildung, dass Tasks aus (Sub-)Tasks zusammengesetzt sein können. Beispielsweise kann die Task PickPart durch die serielle Verknüpfung der Tasks Move (M), GripperAction (G) und Move (M) abgebildet werden. Gemäß [124, 67] kann die TOC nach dem Top-Down (Dekomposition) oder Bottom-Up Prinzip (Komposition) erfolgen.

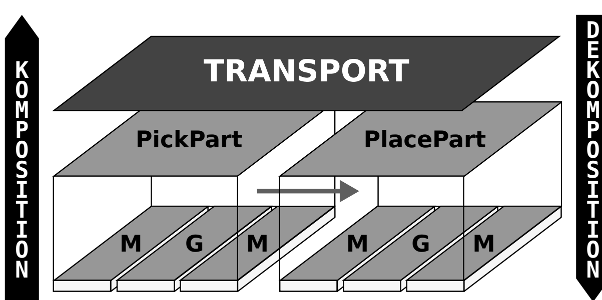


Abbildung 2.14: De-/Komposition eines Problems in mehrere Teilaufgaben

Ein mit Tasks spezifizierter Steuerungsentwurf ist nicht direkt ausführbar, da Tasks eine abstrakte Beschreibung von Arbeitsschritten sind. Eine Task beschreibt nur das *Was* nicht aber das *Wie* oder *Womit* etwas umgesetzt werden soll. Zur Ausführung von Tasks wird eine Transformationsmethode benötigt, wie in Abbildung 2.15 schematisch dargestellt. Auszuführende Tasks werden unter Verwendung eines Weltmodells in Steuerungskommandos für einen realen Prozess transformiert. Als Ergebnis der Transformation liegt ein ausführbarer Steuerungscode in einer für die Applikation spezifischen Sprache vor. In der Robotik kann dies beispielsweise ein Steuerungscode in einer herstellerspezifischen Roboter-

programmiersprache sein. Diese Vorgehensweise entspricht der expliziten Codegenerierung in Unterabschnitt 2.3.2.

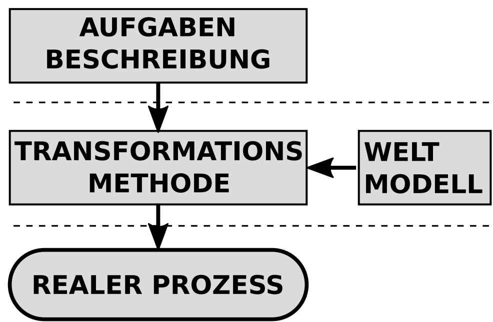


Abbildung 2.15: Schichten einer aufgabenorientierten Steuerung nach Weber [124]

2.4.2 Umsetzung von TOC im SBC-Ansatz

Wie zuvor beschrieben verfolgen TOC basierte Steuerungen klassisch einen kompilierenden Ansatz. Eine aufgabenorientierte Steuerungsspezifikation wird vor der Inbetriebnahme unter der Verwendung eines Weltmodells in eine Menge roboterorientierter Kommandos übersetzt. Neuere Arbeiten wie [67, 102] verstehen diesen Transformationsprozess als Teil der operativen Steuerung. Dies hat den Vorteil, dass auf aktuelle Prozesszustände zurückgegriffen werden kann. Somit lassen sich einfacher reaktive Steuerungen realisieren, welche sich flexibel an aktuelle Prozesszustände anpassen und beispielsweise auf Störungen durch eine alternative Steuerungsvariante reagieren. Da die Aufgabentransformation zur Laufzeit der Steuerung erfolgt, muss sichergestellt sein, dass der Transformationsprozess unter Echtzeitbedingungen ausgeführt werden kann.

Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen. (Scholz [100])

Eine Erweiterung aufgabenorientierter Steuerungsentwürfe stellen parametrierbare Tasks dar [67, 113, 62, 109, 24]. Sie ermöglichen die Zusammenfassung ähnlicher Tasks zu einer gemeinsamen Task. Dadurch wird die Wiederverwendbarkeit von Tasks erhöht und deren Organisation in Repositories erleichtert. Dies ermöglicht die Reduzierung der Entwicklungszeit, Kosten und Fehlerquote.

Abbildung 2.16 zeigt eine Konkretisierung der in Maletzki [67] verbal beschriebenen Umsetzung von TOC im SBC-Framework. Der SBC-Ansatz gemäß Abschnitt 2.3.3 unterscheidet zwischen den drei Komponenten CM, PM und IM. Eine aufgabenorientierte Problembeschreibung wird im CM abgebildet.

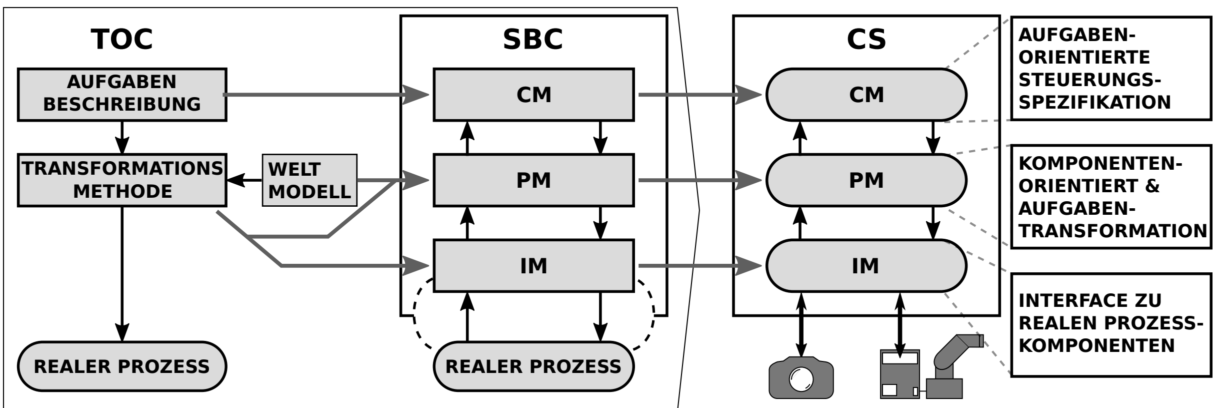


Abbildung 2.16: Umsetzung aufgabenorientierter Steuerungen im SBC-Ansatz

Die Umsetzung der Transformationsmethode erfolgt, wie in Abbildung 2.16 gezeigt, zweigeteilt im PM und IM. Das Weltmodell einer TOC nach Abbildung 2.15 wird durch das PM repräsentiert. Nach Abschnitt 2.3.3 bildet das PM das Wissen über den realen Prozess auf Basis gemessener Prozessgrößen und daraus berechneter Größen ab. Die konkrete Realisierung von TOC mit dem SBC-Ansatz wird im Kapitel 4 gezeigt.

2.5 Interaktionen in Multi-Robotersystemen (MRS)

Die nachfolgenden Ausführungen zu Multi-Robotersystemen basieren grundlegend auf Mataric [69], welche mobile Multi-Robotersysteme betrachtet. Die Analysen lassen sich teilweise auf stationäre Industrieroboter übertragen.

Sind in einer Prozessumgebung mehrere Roboter vorhanden, so steigt die zugrundeliegende Dynamik, da es mehrere Teilnehmer gibt, welche die Umgebung simultan verändern können. Dies macht eine gegenseitige Überwachung einzelner Aktionen der Roboter notwendig. Die Steuerung eines Roboters muss mit Unsicherheiten umgehen können. Diese treten auf, wenn zum Beispiel der Zeitpunkt und die Intention einer Bewegung nicht im Vorfeld bestimmt werden kann. Komplexe Abläufe erfordern, dass sich einzelne Teammitglieder koordinieren müssen, bevor eine Aktionsfolge ausgeführt werden kann. *Koordination* bedeutet die Bestimmung einer Reihenfolge. Wie diese ermittelt wird, kann situationsbedingt variieren. In Roboterteams müssen die Aktionsfolgen koordiniert werden. *Kooperation* bedingt die systematische Ausführung von Aktionen zum gegenseitigen Vorteil. Dies erfordert eine Kommunikation zwischen den Robotersystemen.

Während die Interaktionskonzepte Koordination und Kooperation der mobilen MRS analog für stationäre Gelenkarmroboter gelten, gibt es Unterscheide zu MRS die aus Gelenkarmrobotern bestehen. Im Focus der mobilen MRS liegt zuerst die eigene Lokalisierung. Dann folgt die Verarbeitung der eigenen Wahrnehmung und die Fusion mit den Daten (Position, Wahrnehmungen) der anderen Roboter, um eine gemeinsame Pfadplanung vorzunehmen. Hierfür verfügen mobile Robotersysteme häufig über eine große Anzahl unterschiedlicher Sensoren, mit denen sie ihre Umwelt überwachen. Im Gegensatz zu mobilen Multi-Robotersystemen (MRS) verfügen Industrieroboter in der Regel nicht über so umfangreiche Sensorik und werden ortsfest oder auf einen Portal verfahrbar eingesetzt. Damit ist ihr Arbeitsbereich festgelegt und ihre Position auf Basis einer definierten

Kinematik berechenbar. Unter der Voraussetzung entsprechender Kommunikationsschnittstellen können die Kinematiken mehrerer Roboter verbunden werden, um Bewegungen zu synchronisieren. In diesem Fall wird von einer geometrischen Kopplung gesprochen (Freymann [38]) oder teilweise vom Master/Slave-Systemen (Kosuge und Ishikawa [55]).

2.5.1 Charakteristik von Roboterteams

In diesen Abschnitt werden zunächst Roboterteams als spezifische Form eines MRS charakterisiert. Anschließend werden aufbauend der Literatur sechs Interaktionsklassen als wesentlicher Bestandteil für die nachfolgenden Kapitel abgeleitet.

Der Einsatz eines MRS zur Lösung einer Problemstellung wird auch als Roboterteam bezeichnet. Manche Aufgaben können nicht durch einen Roboter realisiert werden. Beispielsweise kann das Zusammenschweißen zweier Bauteile erfordern, dass diese von jeweils einem Roboter in eine geeignete Position gebracht werden, während ein dritter Roboter sie verschweißt. Es gibt andere Aufgaben, bei denen es nicht grundsätzlich erforderlich ist ein Roboterteam einzusetzen, damit aber Zeit und Kosten eingespart werden können. Ein Beispiel ist das Palettieren von Bauteilen, welche durch ein Förderband in den Arbeitsraum des Roboters gelangen. Hierbei kann im besten Fall durch den Einsatz eines weiteren Roboters der Durchsatz verdoppelt werden. Ein weiteres Anwendungsfeld für den sinnvollen Einsatz mehrerer Roboter ist die sensorische Überwachung von Prozessen. Hierbei können Roboter ihre Messdaten miteinander austauschen und hieraus neue Informationen generieren. Beispielsweise kann ein Roboter ein Kamerasystem führen und damit einen Prozess überwachen. Wird eine Störung detektiert, kann ein weiterer Roboter die Störung beseitigen, während die Prozessüberwachung durch den anderen Roboter fortgesetzt wird. Ein vierter Anwendungsfall ergibt sich aus der größeren Robustheit eines Roboterteams, wenn eine entsprechende Redundanz vorliegt, so dass mehrere Teammitglieder über die selben Fähigkeiten verfügen und einander bei Bedarf ersetzten können. Der Einsatz von MRS ist mit einer Reihe neuer Anforderungen verbunden. Wenn mehrere Roboter in einem Prozess verwendet werden kommt es häufig zu störenden Wechselwirkungen, welche sich physikalisch oder informationell auswirken. Eine wesentliche physikalische Wechselwirkung stellen die Bewegungsbahnen der Roboter dar. Die Verfolgung diametraler Zielstellungen ist eine störende informationelle Wechselwirkung. Demgemäß ist die Kommunikation in MRS ein wesentlicher Aspekt. Es ist festzulegen wer wann kommunizieren darf und welche Informationen auszutauschen sind.

Unterschiedliche Problemstellungen bedingen unterschiedliche Arten von Roboterteams. Eine Möglichkeit diese zu unterscheiden ist eine Klassifikation anhand der Fähigkeiten der Teammitglieder. Das Team kann aus homogenen Teilnehmern bestehen, die alle über die gleichen Fähigkeiten verfügen oder inhomogen aufgebaut werden. Bei homogenen Teams kann jeder Teilnehmer einen anderen Teilnehmer komplett ersetzen, falls dieser nicht zur Verfügung steht oder ausgefallen ist.

Eine weitere Klassifikation von Roboterteams kann anhand der Koordinationsstrategie erfolgen. Es werden drei Arten unterschieden. Im einfachsten Fall existieren die Teilnehmer eines Roboterteams nebeneinander und kommunizieren nicht miteinander. Ein Roboter kann einen anderen Roboter durch seine Sensoren erkennen und als eine dynamische Prozesskomponente interpretieren. Ein anderer Roboter kann den eigenen Arbeitsraum

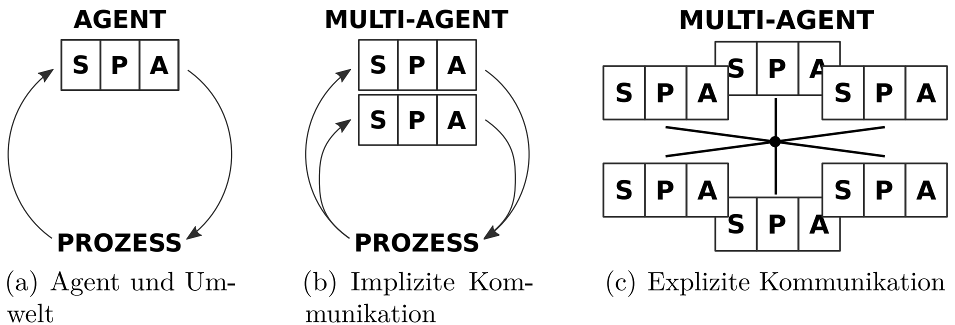
kurzzeitig betreten und damit die Ausführung einer geplanten Aktion verzögern. *Lose koordinierte Robotersysteme* können als Team aufgefasst werden, wenn sie als Gruppe an einer gemeinsamen Aufgabenstellung arbeiten. Eine weitere Klasse bilden *grob gekoppelte Systeme*. Hierbei erkennen die Roboter einander und können einfache koordinierte Aktionen, wie beispielsweise das gemeinsame Verschieben eines Bauteils, miteinander aushandeln. Ein wichtiges Merkmal ist, dass die Roboter nicht grundsätzlich voneinander abhängig sind. Das heißt, dass ein Teammitglied ausfallen kann und die Arbeitsaufgabe dennoch, eventuell mit zeitlichen Verzug, ausgeführt werden kann. Zur dritten Klasse, der *eng gekoppelten Systeme*, zählen alle Roboterteams, bei denen die Leistungsfähigkeit eines Roboters stark von den Fähigkeiten mindestens eines anderen Roboters abhängt. Die Koordination der Teammitglieder erfordert viel Kommunikation. Weiterhin sind die Teammitglieder nicht identisch und somit nicht untereinander austauschbar. Solche Roboterteams verfügen über eine hohe Leistungsfähigkeit und sind auf Grund fehlender Redundanz weniger robust gegenüber Störungen.

2.5.2 Interaktionsklassen von Roboterteams

Im Kontext dieser Arbeit ist ein MRS ein Roboterteam, bestehend aus mehreren Industrierobotern mit lokalem Steuerungsrechnern als operative Controller und einem übergeordneten Steuerungsrechner mit einem Supervisory-Control-Programm (Freymann [38]). Ein solches Roboterteam weist gegenüber vernetzten Robotern mit unabhängigen lokalen Steuerungsrechnern verschiedene Vorteile auf. In der übergeordneten Supervisory-Control werden die relevanten Sensorinformationen und Zustandsgrößen aller Roboter abgebildet und ein gemeinsames Umweltmodell geschaffen. Das gemeinsame Umweltmodell vereinfacht die Abbildung der gegenseitigen Wechselwirkungen der Roboter im Vergleich zu rein zentral aufgebauten Steuerungen.

Lüth [64] führt an, dass selbständig handelnden Systeme, welche kooperativ und koordiniert zusammenwirken, als Agenten bezeichnet werden. In diesem Zusammenhang stellen Roboter technische Agenten dar, welche nicht nur informationstechnisch, sondern auch physikalisch miteinander wechselwirken. Für zielgerichtete Wechselwirkungen müssen sich die technischen Agenten zeitlich und örtlich abstimmen. Die Wechselwirkungen zwischen den Agenten in einem MRS werden als Interaktionen bezeichnet [64].

Wie im Abschnitt 2.1 zur Steuerungsarchitektur diskutiert wurde, muss jeder Agent seinen Prozess bzw. seine Umwelt überwachen, um diese zielgerichtet zu beeinflussen. Dies wird in Abbildung 2.17 (a) schematisch gezeigt. Der Agent besteht aus drei wesentlichen Komponenten: Sensorik (S), Informationsverarbeitung (Processing, P) und Aktorik (A). Die Komponente S überwacht den Prozess oder die Umgebung. Die Komponente P wertet die gesammelten Informationen aus und trifft Entscheidungen, wie die Umgebung durch die Komponente A zu manipulieren ist. Befinden sich in einer Umgebung mehrere Agenten, wie im Falle eines MRS, so können diese die Änderungen eines anderen Agenten an der Umgebung überwachen. Dies gleicht einer impliziten Kommunikation und wird in Abbildung 2.17 (b) dargestellt. Neben der impliziten Kommunikation können Agenten eines MRS auch explizit, d.h ohne den Umweg über die Umgebung, miteinander kommunizieren (vgl. Teilbild (c)).



(a) Agent und Umwelt

(b) Implizite Kommunikation

(c) Explizite Kommunikation

Abbildung 2.17: Implizite und explizite Kommunikation von MRS nach Behnke [8]

Abbildung 2.18 zeigt den Zusammenhang zwischen einem Verteilten System (VS), einem MRS und einem Multi-Agentensystem (MAS). Coulouris [23] und Hammerschall [46] definieren ein VS als ein System aus Hardware- und Softwarekomponenten mit vernetzten Computern, welche miteinander über den Austausch von Nachrichten kommunizieren. In diesem Sinne bildet ein MRS mit einer Supervisory-Control (Steuerungscomputer) und operativen Roboter-Controllern (SRS) ein VS. Zusätzlich sind im MRS sie Aktuatoren mitzubetrachten. Beinhaltet das System weitere Agenten gemäß der obigen Definition, wie in Abbildung 2.18 gezeigt, handelt es sich um ein MAS. Demgemäß ist ein MAS, ein Verteiltes System (VS), welches durch mehrere Agenten geformt wird, die nebenläufig und autonom agieren. Wie Tolk [116] diskutiert, sind autonome Robotersysteme und intelligente Softwareagenten topologisch und konzeptionell verwandt. Dies bedeutet, dass Erkenntnisse beider Anwendungsgebiete aufeinander übertragen werden können.

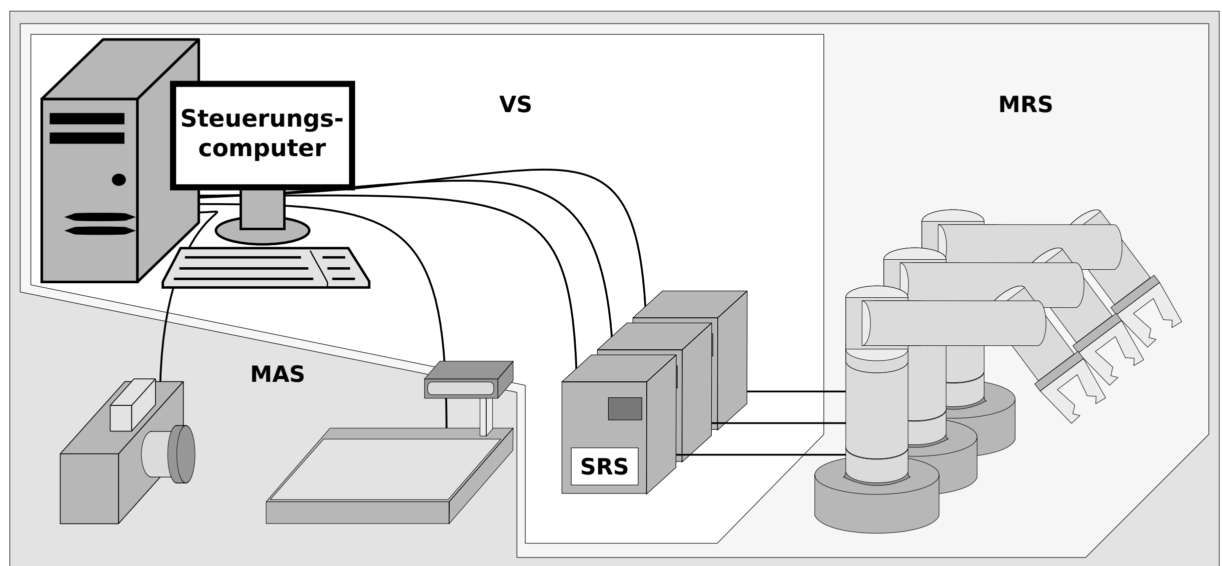


Abbildung 2.18: Zusammenhang von VS, MRS und MAS

Nach Ritter [112] beschäftigt sich die bisherige Forschung vorwiegend mit den internen Abläufen eines Agenten. Hierunter versteht Ritter die Definition von Zielen, die Repräsentation von Wissen und die logische Entscheidungsfindung. Er merkt an, dass für bestimmte Anwendungsfälle eine Kooperation mehrerer Agenten notwendig ist. Bei der Entwicklung von MAS ist neben der Kommunikation oft die Koordination der Agenten von

Bedeutung. In einem MRS findet Interaktion statt, wenn mehrere SRS durch eine Menge wechselseitiger Aktionen in Beziehung zueinander gebracht werden. Die Interaktionen entstehen aus Aktionen, die Einfluss auf das weitere Verhalten der einzelnen SRS haben.

In der Regel dienen Interaktionen in MRS dem Ziel, die Leistungsfähigkeit eines Roboterteams zu erhöhen. Im optimalen Fall erfolgt eine Interaktion zum allerseitigen Vorteil. Jedoch kann eine Interaktion eine temporäre Zuordnung einer limitierten Ressource erfordern, welche dann einem anderen Agenten nicht mehr zur Verfügung steht. Im Fall eines MRS sollte die Steuerung proaktiv die Folgen einer Interaktion abschätzen, um den bestmöglichen Zeitpunkt für eine Wechselwirkung zu bestimmen. Weiterhin muss eine Steuerung festlegen, welche ihrer Agenten in Interaktion treten. Die Entscheidungsfindung kann stark von den Fähigkeiten und der Leistungsfähigkeit der aktuell zur Verfügung stehenden Agenten abhängen. Sollen beispielsweise zwei Roboter ein schweres Bauteil gemeinsam von einer Position an eine andere Position transportieren, so muss sich das Bauteil im gemeinsamen Arbeitsraum der Roboter befinden. Gleichermaßen müssen beide Roboter zu diesem Zeitpunkt zur Verfügung stehen.

Interaktionen von Industrierobotern in MRS lassen sich, basierend auf der allgemeinen Klassifizierung in Lüth [63], in sechs Klassen einteilen. Zur besseren Veranschaulichung werden diese nachfolgend am Beispiel eines Transportproblems diskutiert. Beim Beispiel sollen Bauteile von einem Input-Buffer (IB) zu einem Output-Buffer (OB) transportiert werden. Abbildung 2.19 zeigt schematisch die Struktur des Transportproblems mit einen SRS. Ausgehend von dem SRS-Ansatz zeigt Abbildung 2.20 die grundlegenden Strukturen des Transportproblems bezogen auf die sechs Interaktionsklassen eines MRS.

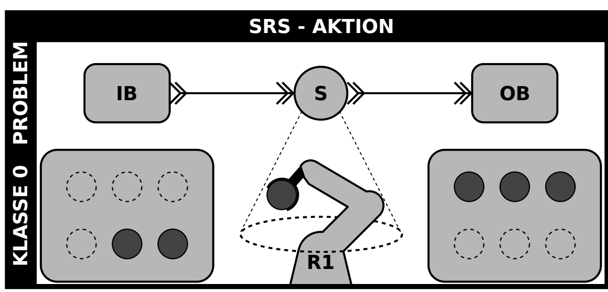


Abbildung 2.19: Transportproblem mit einen SRS

Klasse 0: Die Ausgangssituation in Abbildung 2.19 zeigt ein SRS und beinhaltet keine Interaktion. Im Kontext des Transportproblems kann der Roboter R1 als ein Server S mit der Kapazität eins verstanden werden. Dieser hat die Aufgabe jeweils ein Teil vom IB zum OB zu transportieren. Es ist nur eine Teileart zu bewegen. Das Werkzeug des Roboters ist der Teileart angepasst. Die Aufgabe ist beendet, wenn alle Bauteile vom Eingangspuffer zum Ausgangspuffer transportiert wurden.

Zur Betrachtung der Interaktionsklassen in einem MRS wird das Transportproblem schrittweise adaptiert. Die Komplexität der Transportaufgabe nimmt von Klasse 1 bis 6 zu.

Interaktionsklassen:

Klasse 1: Es wird ein MRS bestehend aus zwei Robotern (R1, R2) mit voneinander getrennten Arbeitsräumen betrachtet. Beide Roboter besitzen identische Werkzeuge und Fähigkeiten. Es ist kein Austausch von Informationen zwischen den Robotern erforderlich.

Die Interaktion bezieht sich auf die parallele Lösung einer Problemstellung durch zwei Roboter. Die Problemstellung ist gelöst, wenn beide Roboter ihre Aufgaben abgearbeitet haben. Unter optimalen Bedingungen kann die Bearbeitungszeit gegenüber Klasse 0 (dem SRS) halbiert werden.

Klasse 2: Analog zu Klasse 1, jedoch erweitert um eine neue Teileart, die eine Modifikation eines Roboters bedingt. Dies kann z.B. einen Austausch des Greifers erfordern. Die zwei Teilearten sind räumlich voneinander getrennt. Jeder Roboter transportiert nur eine Teileart. Durch die Trennung der Arbeitsräume ist keine Abstimmung der Roboter und somit keine Kommunikation und Koordination erforderlich. Im Gegensatz zur Klasse 1 sind die Roboter nicht gegenseitig austauschbar.

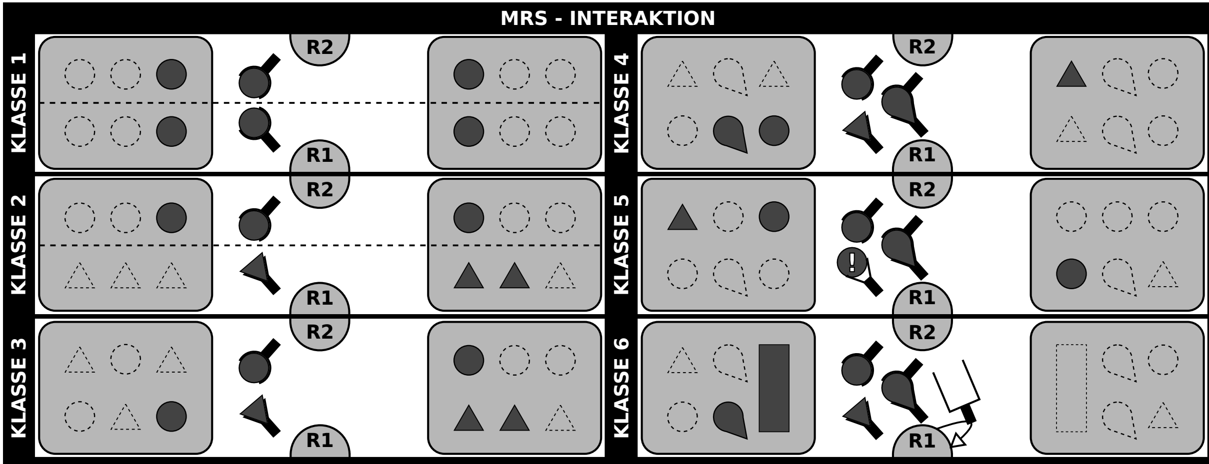


Abbildung 2.20: Schematische Darstellung des Transportproblems für die Interaktionsklassen eines MRS in Anlehnung an die Klassifikation in Lüth [63]

Klasse 3: Es werden die zwei Teilearten der Klasse 2 verwendet. Jedoch sind diese nicht mehr räumlich voneinander getrennt. Der Arbeitsbereich der Roboter überlappt sich folglich, sodass eine Koordination der Bewegungsfolgen notwendig wird, um Kollisionen zu vermeiden. Hierzu ist ein Informationsaustausch, eine Kommunikation, erforderlich, um die Bewegungsabläufe der Roboter zu koordinieren.

Klasse 4: Durch die Einführung einer weiteren Teileart, welche nur durch eine unmittelbare Zusammenarbeit beider Roboter bewegt werden kann, erhöht sich der Kommunikations- und Koordinationsaufwand. Die Transportaufgabe kann nur beendet werden, wenn beide Robotersysteme zusammenarbeiten, um die neue Teileart zu transportieren. Hierbei muss die Steuerung den Zeitpunkt der gemeinsamen Interaktion planen. Erfolgt keine zeitliche Abstimmung des Transports der neuen Teileart, kann ein Roboter durch unnötiges Warten blockiert werden.

Klasse 5: Es gelten die Anforderungen der Klasse 4. Zusätzlich kann jeder Roboter die Teilearten des anderen Roboters händeln, um den Partner in einer Überlastsituation zu unterstützen. Es folgt keine prinzipiell neue Anforderung hinsichtlich der Kommunikation und Koordination.

Klasse 6: Es ist eine weitere Teileart zu transportieren, die nicht durch die beiden Roboter gehandhabt werden kann. Es folgt die Notwendigkeit einer neuen Interaktion, zur dynamischen Funktionserweiterung des MRS. Diese kann realisiert werden durch einen

Werkzeugwechsel oder die temporäre Integration eines weiteren Roboters in das MRS.

Zusammenfassung und Verallgemeinerung: Tabelle 2.1 fasst die wichtigsten Merkmale der sechs Klassen zusammen. Bei Klasse 1 sind alle Roboter vom selben Typ und verfügen über identische Fähigkeiten. Eine gegenseitige Arbeitsraumüberwachung ist nicht erforderlich. Bei Klasse 2 unterscheiden sich die Robotertypen, sodass eine Spezialisierung der Fähigkeiten vorliegen kann. Wie bei Klasse 1, sind die Arbeitsräume voneinander getrennt. Klasse 1 und 2 erfordern keine explizite Kommunikation. Bei Klasse 3 ist die Trennung der Arbeitsräume aufgehoben. Der Zugriff auf gemeinsame Ressourcen muss koordiniert werden, um Kollisionen zu vermeiden. Es muss eine implizite oder explizite Kommunikation erfolgen. Bei Klasse 4 sind einige Aufgaben nur im Verbund mehrerer Roboter zu lösen. Dies erfordert eine Kooperation zwischen den Robotern und die Realisierung entsprechender Synchronisationsmechanismen. Das heißt, dass sowohl die Bewegungstrajektorien, als auch der Zeitpunkt einer Kooperation zur Laufzeit ausgehandelt werden muss. Bei Klasse 5 übernehmen die Roboter "ungeeignete" Aufgaben, falls sie über freie Kapazitäten verfügen. Die Roboter unterstützen einander, um die Leistungsfähigkeit des Teams zu erhöhen. Klasse 6 diskutiert den Fall, dass sich ein Roboterteam zur Laufzeit der Steuerung verändern kann. Die Fähigkeiten des Roboterteams werden situationsbedingt adaptiert.

Tabelle 2.1: Wesentliche Merkmale der Interaktionsklassen

Klasse	**Merkmal**
1	gleichartige Roboter und Trennung der Arbeitsräume; keine explizite Kommunikation
2	ungleichartige Roboter und Trennung der Arbeitsräume; keine explizite Kommunikation
3	Koordination bzgl. verfügbarer Ressourcen durch Kommunikation
4	Kooperation der Roboter mittels Synchronisationsmechanismen
5	implizite Aufgabenumverteilung (gegenseitige Unterstützung)
6	temporäre Teamreorganisation (Anpassung der Leistungsfähigkeit an Situation)

2.6 Zusammenfassung

In diesem Kapitel wurden zuerst verschiedene Steuerungsparadigmen zur Entwicklung von Robotersteuerungen vorgestellt. Weiterhin erfolgte eine kurze Betrachtung von möglicher Middleware, welche eine Integration unterschiedlicher Robotersysteme ermöglicht. Anschließend wurden unterschiedliche Methoden zur Online- und Offline-Programmierung von Robotersystemen dargelegt. Hierbei zeigte sich, dass die Anwendung eines Vorgehensmodells im Rahmen der Steuerungsentwicklung sinnvoll ist. Es wurden drei aus der Literatur bekannte Vorgehensmodelle im Kontext der Robotersteuerungsentwicklung analysiert.

Als Ergebnis der Analyse wurde die aufgabenorientierte Steuerungsentwicklung nach dem SBC-Vorgehensmodell als Basis für die weiteren Untersuchungen ausgewählt. Hinsichtlich der roboterorientierten Middleware wurde sich für die MATLAB-basierte RCV-Toolbox entschieden, da diese eine herstellerunabhängige Entwicklung und im Vergleich zu roboterorientierten Sprachen eine effiziente Programmentwicklung unterstützt.

Anschließend wurde der Stand der Technik hinsichtlich der Entwicklung von aufgabenorientierten Steuerungen für SRS nach dem SBC-Ansatz untersucht. Dabei zeigte sich, dass es bisher diesbezüglich keine Arbeiten zur Steuerungsentwicklung für MRS gibt.

Abschließend erfolgte auf Basis der Literatur eine Charakterisierung von Roboterteams, die als spezifische MRS in dieser Arbeit betrachtet werden. Der Schwerpunkt wurde dabei auf die Interaktion gelegt. Ausgehend von der Literatur wurden als Basis für die weiteren Betrachtungen sechs Interaktionsklassen spezifiziert.

3 Ereignisdiskrete Simulation und Steuerung mit dem DEVS-Formalismus

Im vorangegangenen Kapitel wurde in einem Unterabschnitt auf Vorgehensmodelle und Frameworks zur zielgerichteten und systematischen Umsetzung automatisierungstechnischer Problemstellungen eingegangen. Dabei wurde explizit die durchgängige Nutzung von M&S-Methoden bei der Steuerungsentwicklung diskutiert. Neben der Forderung nach Durchgängigkeit steht die Anforderung der Wiederverwendbarkeit beziehungsweise Erweiterungsfähigkeit von Modellen – möglichst von der frühen Entwurfsphase bis in die Betriebsphase von Steuerungen. Dazu wird im ersten Unterabschnitt die Relation zwischen ereignisdiskreter Simulation und ereignisdiskreter Steuerung betrachtet.

In diesem Kapitel wird eingehend untersucht, ob der Discrete-Event-System-Specification (DEVS)-Formalismus die Anforderung einer durchgängigen Nutzung von Modellen von der Entwurfsphase bis in den operativen Steuerungsbetrieb erfüllt. Der originäre DEVS-Formalismus von Zeigler [132, 133] wird heute als Classic-DEVS bezeichnet [131]. Aufbauend auf Classic-DEVS wurde von Chow [19] der Parallel-DEVS-Ansatz eingeführt. In beiden Formalismen fehlen die für Steuerungen notwendige Unterstützung von Echtzeitfähigkeit sowie Schnittstellen zur Prozessanbindung.

Basierend auf Classic-DEVS und Parallel-DEVS wurden verschiedene Erweiterungen des DEVS-Formalismus zur Umsetzung von Steuerungen entwickelt. In diesem Kapitel werden im Abschnitt 3.3 bekannte DEVS-Formalismen analysiert, die Erweiterungen bezüglich Echtzeitfähigkeit und Schnittstellen zur Prozessanbindung einführen. Zum Verständnis dieser Formalismen werden zuvor im Abschnitt 3.2 wesentliche Grundlagen von Classic- und Parallel-DEVS eingeführt. Die Analyse der DEVS-Formalismen soll zeigen, ob ein Formalismus die gestellte Anforderung der durchgängigen modellbasierten Entwicklung von Steuerungen erfüllt.

Die mengentheoretische Definition der DEVS-Formalismen ist für Nutzer oft nicht intuitiv verständlich. Aus diesem Grund wurden unterschiedliche graphische Notationen für DEVS entwickelt (Praehofer und Pree [88], Zinoviev [137], Bonaventura et al. [13]). Aufbauend auf der graphischen Notation von Song und Kim [111] wurde in dieser Arbeit eine erweiterte Notation entwickelt, welche im Abschnitt 3.2.3 eingeführt wird. Die Erweiterungen orientieren sich an Elementen von Statecharts. Die graphische Notation wird in diesem Kapitel im Rahmen der Analyse der DEVS-Formalismen benutzt. Ein Teil der Erweiterungen wird erst in späteren Kapiteln dieser Arbeit verwendet.

3.1 Beziehung zwischen ereignisdiskreter Simulation und ereignisdiskreter Steuerung

Die VDI-Richtlinie 3633-1 [32] charakterisiert Simulation, analog zu Shannon [106], als ein Verfahren zur Nachbildung des dynamischen Verhaltens realer oder gedachter Systeme, um mittels gezielter Experimente zu Erkenntnissen zu gelangen, die auf die Realität übertragbar sind. Die ereignisdiskrete Simulation (Discrete-Event-Simulation (DES)) ist auf eine bestimmte Sichtweise dynamischer Systeme fokussiert. Bei der DES wird die Systemdynamik als eine diskrete Sequenz von Ereignissen über der Zeit betrachtet. Jedes Ereignis tritt zu einem bestimmten Zeitpunkt auf, besitzt eine zeitlose Dauer und verändert den Zustand des Systems. Zwischen aufeinanderfolgenden Ereignissen ändert sich der Systemzustand nicht. Daraus folgt nach Zeigler et al. [132, 131, 135] eine Systemdynamik in Form einer Menge von Ereignissen $e \in \{[e_1, e_2, e_3, ..., e_n]\}$ und einer Menge sequentieller Zustände $q \in Q$:

$Q = \{[(s_0, \tau_0), (s_1, \tau_1), ..., (s_n, \tau_n)]\}$ mit:
$s_n \in S$ Menge der Zustände
$\tau_n \in \mathbb{R}^+$ Verweildauer

Diese Sichtweise korrespondiert mit der Charakterisierung ereignisdiskreter Steuerungen gemäß Zander [129] und Hrúz et al. [49]. Eine ereignisdiskrete Steuerung ist ebenfalls gekennzeichnet durch eine Folge von Zustand-Verweildauer-Paaren $q \in Q$. Die Zustandsänderungen werden durch diskrete Ereignisse ausgelöst, d.h. es gilt: $e_1(q_0) \rightarrow q_1, e_2(q_1) \rightarrow q_2, ..., e_n(q_{n-1}) \rightarrow q_n$. Dabei sei erwähnt, dass bei nicht zeitbehafteten Steuerungen die Verweildauern $\tau \in \mathbb{R}^+$ der Zustände nicht betrachtet werden.

Robotersteuerungen auf der Ebene der Handlungsplannung (vgl. Abb. 2.5) sind typische Vertreter ereignisdiskreter Steuerungen. Kontinuierliche Systemdynamik wird oft mittels Quantisierung in ereignisdiskrete Dynamik transformiert, vgl. Abel und Bolling [1].

Die von Zeigler 1976 eingeführte DEVS ist ein system-theoretisch-basierter Formalismus zur Spezifikation und Simulation ereignisdiskreter Systeme [132]. Der Formalismus wurde kontinuierlich weiterentwickelt [131, 135]. Da im Kontext der Arbeit ausschließlich Aspekte ereignisdiskreter Steuerungen zu untersuchen sind, wird nachfolgend der DEVS-Formalismus nur aus Sicht ereignisdiskreter Systeme betrachtet, allerdings unter Einbeziehung von Erweiterungen hinsichtlich der Integration mit realen Prozessen. Letzteres bezieht sich auf Aspekte der Echtzeitfähigkeit und Prozessanbindung.

3.2 DEVS-Formalismen

In diesem Abschnitt wird der DEVS Formalismus auf Basis von [131] eingeführt. Alle im Rahmen dieser Arbeit relevanten DEVS-Ausprägungen werden bezüglich der Modellspezifikation und den grundlegenden Simulationsalgorithmen erläutert. Im Fokus der Betrachtungen steht der Einsatz von DEVS für Echtzeitanwendungen im Rahmen ereignisdiskreter Robotersteuerungen.

Der DEVS-Formalismus definiert eine systemtheoretisch-basierte Modellspezifikation

und dazugehörige Abarbeitungsalgorithmen, eine operationelle Semantik. Bei der Modellspezifikation werden zwei DEVS-Systemtypen unterschieden. Das dynamische Verhalten wird mit atomaren (Atomic) DEVS-Modellen beschrieben. Daneben gibt es die gekoppelten (Coupled) DEVS-Modelle, welche Kompositionen aus atomaren beziehungsweise weiteren gekoppelten DEVS-Modellen definieren. Somit kann jedes Coupled-DEVS wiederum Bestandteil eines anderen Coupled-DEVS sein, sodass hierachische Modelle entstehen. D.h., gekoppelte DEVS Systeme beschreiben Mengen von Modellen und deren Kopplungsbeziehungen. Das durch diese Komposition entstehende Coupled-DEVS-Modell verfügt über die selben Eigenschaften wie ein Atomic-DEVS-Modell. Diese Eigenschaft wird als *Clossure-Unter-Coupling* bezeichnet.

Die Herangehensweise von DEVS ist, die Modellspezifikation streng von den Simulationsalgorithmen zu trennen. Ein Beispiel eines ausführbaren *Computermodells* zeigt Abbildung 3.1. Wie in der Abbildung zu sehen ist, sind bei DEVS auch die Simulationsalgorithmen hierarchisch aufgebaut. Hierbei wird jedem atomaren DEVS-Modell ein Simulator und jedem gekoppelten DEVS-Modell ein Koordinator zugeordnet. Zusätzlich wird dem in der Hierarchie höchsten Koordinator ein Root-Koordinator zugeordnet. Dieser ist für die Initialisierung und Ausführung der gesamten Simulation verantwortlich. Der Informationsaustausch zwischen Koordinatoren und Simulatoren basiert auf einem festen Nachrichtenprotokoll.

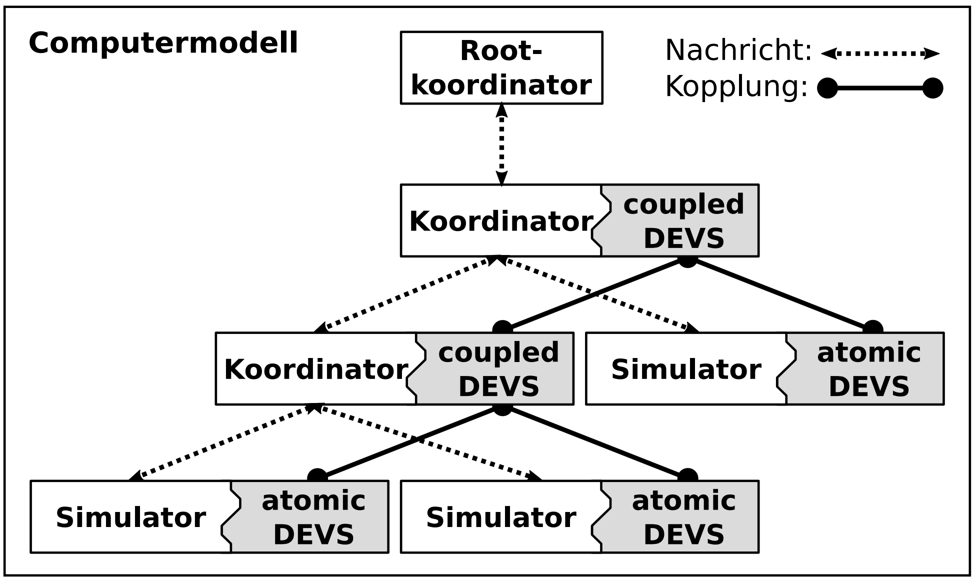


Abbildung 3.1: Struktur eines DEVS-Computermodells. Zuordnung von Simulationsalgorithmus (linke Box) zu Modellspezifikation (rechte Box).

3.2.1 Classic-DEVS-Formalismus und die Erweiterung DEVS mit Ports

Der Classic-DEVS-Formalismus, wurde 1976 von Zeigler [132] eingeführt und ist bis heute die Grundlage sämtlicher Weiterentwicklungen. Wie bei allen DEVS Formalismen erfolgt eine strikte Trennung zwischen Modellspezifikation und den Simulationsalgorithmen.

Das Atomic-Classic-DEVS ist wie in Tabelle 3.1 definiert:

Tabelle 3.1: Atomic-Classic-DEVS

$\mathbf{DEVS} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \lambda, \mathbf{ta})$		
X		Menge der Eingangsereignisse
S		Menge der Zustandswerte
Y		Menge der Ausgangsereignisse
$\delta_{\mathbf{int}}$	$\delta_{\text{int}} : \mathrm{S} \to \mathrm{S}$	interne Zustandsüberführungsfunktion
$\delta_{\mathbf{ext}}$	$\delta_{\text{ext}} : \mathrm{Q} \times \mathrm{X} \to \mathrm{S}$	externe Zustandsüberführungsfunktion
	$\mathrm{Q} = \{(\mathrm{s}, \mathrm{e}) \vert \mathrm{s} \in \mathrm{S}, 0 \leq \mathrm{e} \leq \mathrm{ta}(\mathrm{s})\}$	Menge totaler Zustände
	e (elapsed time)	verstrichene Zeit seit letztem Ereignis
λ	$\lambda : \mathrm{S} \to \mathrm{Y}$	Ausgabefunktion
ta	$\mathrm{ta} : \mathrm{S} \to \mathbb{R}^{+}_{0,\infty}$	Zeitfortschrittsfunktion

Das dynamische Verhalten eines Atomic-Classic-DEVS ist in Abbildung 3.2 schematisch dargestellt und wird mithilfe von vier Funktionen spezifiziert. Jedes Atomic-Classic-DEVS besitzt ein *Gedächnis*, in Form einer Zustandmenge S. Mittels der Zeitfortschrittsfunktion ta(s) wird aus dem aktuellen Zustand $\mathrm{s} \in \mathrm{S}$ die Zeitspanne bis zum nächsten internen Ereignis bestimmt. Sobald diese Zeitspanne abgelaufen ist, wird die Ausgabefunktion $\lambda(\mathrm{s})$ ausgeführt und auf Grundlage des aktuellen Zustands $\mathrm{s} \in \mathrm{S}$ ein Ausgangsereignis $\mathrm{y} \in \mathrm{Y}$ erzeugt. Danach wird die interne Zustandsüberführungsfunktion $\delta_{\text{int}}(\mathrm{s})$ ausgelöst, welche auf Grundlage des aktuellen Zustands s den nachfolgenden Zustand $\mathrm{s}‘ \in \mathrm{S}$ ermittelt. Sobald ein externes Ereignis $\mathrm{x} \in \mathrm{X}$ anliegt und sich das System im totalen Zustand $\mathrm{q} \in \mathrm{Q}$ mit $\mathrm{e} < \mathrm{ta}(\mathrm{s})$ befindet, wird die externe Zustandsüberführungsfunktion $\delta_{\text{ext}}(\mathrm{s}, \mathrm{e}, \mathrm{x})$ ausgeführt. Diese bestimmt auf Grundlage des aktuellen Zustands s, der verstrichenen Zeit e seit dem letzten Zustandsübergang und dem aktuellen externen Ereignis x den nachfolgenden Zustand $\mathrm{s}‘ \in \mathrm{S}$.

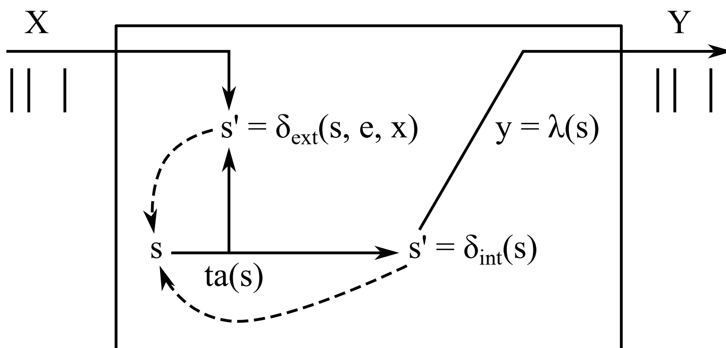


Abbildung 3.2: Dynamik eines Atomic-Classic-DEVS nach Zeigler [131]

Eine erste Weiterentwicklung des Atomic-Classic-DEVS-Formalismus stellt die Erweiterung durch multiple Ein- und Ausgangsschnittstellen (Ports) dar. Diese vereinfachen oft den Modellierungsprozess. Ein Atomic-DEVS mit Ports ist gemäß Tabelle 3.2 definiert. In diesem Kontext stellt p die Bezeichnung eines konkreten Ports und v den aktuellen Wert dar.

Tabelle 3.2: Atomic-Classic-DEVS mit Ports

$\mathbf{DEVS-Ports} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \lambda, \mathbf{ta})$		
$\mathbf{S}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \lambda, \mathbf{ta}$		analog Classic-DEVS
X	$X = \{(p,v) \mid p \in \text{IPorts}, v \in X_p\}$	Menge der Eingangswert-Port-Paare
Y	$Y = \{(p,v) \mid p \in \text{OPorts}, v \in Y_p\}$	Menge aller Ausgangswert-Port-Paare

Ein Classic-Coupled-DEVS (auch DEVS-Network genannt, DEVN) ist in Tabelle 3.3 definiert. Die Unterscheidung zwischen dem Classic-DEVS Formalismus mit und ohne Ports erfolgt lediglich durch die unterschiedliche Spezifikation der Mengen X und Y.

Die *Select-Funktion* bestimmt die Abarbeitungsreihenfolge von simultanen internen Ereignissen innerhalb eines *Coupled-Classic-DEVS*. Dabei werden alle Komponenten mit zeitgleichen internen Ereigniszeitpunkten als *Imminents* bezeichnet.

Tabelle 3.3: Coupled-DEVS auch genannt DEVS-Network (DEVN)

$\mathbf{DEVN} = (\mathbf{X}, \mathbf{Y}, \mathbf{D}, \{\mathbf{M_d}\}, \{\mathbf{I_d}\}, \{\mathbf{Z_{i,d}}\}, \mathbf{Select})$			
X,Y			analog atomic-DEVS
D			Indexmenge der Subkomponenten
$\mathbf{M_d}$	Modellspezifikation der Komponenten für $d \in D$ mit		
	$Md = \{X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta\}$		atomic-DEVS oder coupled-DEVS
$\mathbf{I_d}$	ist die Menge aller Einwirkungen auf $d \in D$ mit $I_d \subseteq D \cup \{N\}, d \notin I_d$		
$\mathbf{Z_{i,d}}$	Menge der $i - to - d$ Kopplungsbeziehungen mit $i \in I_d$, für die gilt:		
	$Z_{i,d} : X \to X_d$,	wenn $i = N$	externe Eingangskopplung
	$Z_{i,d} : Y_i \to Y$,	wenn $d = N$	externe Ausgangskopplung
	$Z_{i,d} : Y_i \to X_d$,	wenn $d \neq N$ und $i \neq N$	interne Kopplung
Select	wählt *Imminent* mit höchster Priorität		Konfliktlösungsfunktion

DEVS-Modelle werden immer ereignisorientiert ausgeführt. Dazu wird wie in Abbildung 3.1 jedem Atomic-DEVS ein Simulator und jedem Coupled-DEVS ein Koordinator zugeordnet. Ein Simulator stellt die Ablaufsteuerung für ein Atomic-DEVS-Modell bereit und verarbeitet dessen Modellspezifikation. Die Ablaufsteuerung eines Coupled-DEVS-Modells ist dagegen im Koordinator implementiert. Weiterhin ist allen Koordinatoren und Simulatoren ein Hauptkoordinator (root-coordinator) übergeordnet, welcher solange die Simulationszeit fortschaltet, bis ein definierter Abbruchzeitpunkt oder ein Abbruchereignis eintritt. Wie in Tabelle 3.4 dargestellt ist, kommunizieren die einzelnen Koordinatoren und Simulatoren mittels vier unterschiedlicher Nachrichtentypen miteinander.

Tabelle 3.4: Nachrichtenprotokoll der Classic-DEVS Simulationsalgorithmen

Initialisierungsnachricht (init-message)
Die Nachricht wird einmalig zu Beginn der Simulation top-down vom Root-Koordinator ausgehend an alle Simulatoren und Koordinatoren versendet. Erst im Anschluß beginnt die eigentliche Simulationsphase.
Eingangsnachricht (x-message)
Die Nachricht wird für die Übertragung von Ereignissen $x \in X$ genutzt und führt beim Empfänger zu einem externen Ereignis. Innerhalb der Koordinatoren werden sämtliche Eingangsereignisse gemäß den Kopplungsbeziehungen in der Modellspezifikation des zugeordneten coupled-DEVS weitergeleitet. Im Gegensatz dazu ruft ein Simulator die externe Zustandsüberführungsfunktion sowie die Zeitfortschrittsfunktion auf.
Zustandsnachricht (star-message)
Die Simulationsphase wird durch das Senden einer Zustandsnachricht vom Hauptkoordinator an den obersten Koordinator eingeleitet. Sie veranlasst jeden Koordinator, in Abhängigkeit von der Select-Funktion des zugeordneten DEVN, genau eine Imminent-Komponente des DEVN auszuwählen und sendet an dessen Koordinator oder Simulator die Zustandsnachricht weiter. Wenn ein Simulator eine Zustandsnachricht erhält, dann ruft dieser neben der Ausgabefunktion auch die interne Zustandsüberführungsfunktion sowie die Zeitfortschrittsfunktion seines zugeordneten atomic-DEVS auf.
Ausgangsnachricht (y-message)
Die Nachricht wird zur Übertragung von Ausgangsereignissen genutzt und ausschließlich von Simulatoren und Koordinatoren an den ihnen übergeordneten Koordinator versendet. Dieser transformiert die Ausgangsereignisse gemäß den Kopplungsbeziehungen seiner zugeordneten DEVS-Modelle in Eingangsereignisse anderer DEVS-Modelle.

Der Simulationsalgorithmus für den Simulator eines Atomic-DEVS-Modells ist in Listing 3.1 dargestellt. Listing 3.2 zeigt den Algorithmus eines Koordinators eines Coupled-DEVS. An dieser Stelle soll explizit die Verarbeitung simultaner Ereignisse betrachtet werden. Aufgrund des Nachrichtenprotokolls wird beim Classic-DEVS-Formalismus bei zeitgleichen internen Ereignissen:

- erst die höchstpriorisierte Komponente ausgewählt, welche Ausgangsereignisse versendet, einen internen Zustandsübergang ausführt und ihren nächsten Ereigniszeitpunkt einplant
- dann werden Ausgangsereignisse in Eingangsereignisse anderer Komponenten transformiert, worauf diese *externe Zustandsüberführungen* ausführen und ihren nächsten Ereigniszeitpunkt neu einplanen. Somit kann nach Ausführung eines externen Ereignisses zum gleichen Zeitpunkt für ein Atomic-DEVS ein internes Ereignis neu- oder wieder eingeplant werden.

Listing 3.1: DEVS-Simulator nach [131]

```
1  Variables:
2    parent // parent coordinator
3    tL  // time of last event
4    tN  // time of next event
5  when receive init−message(t)
6    tL ← t;
7    tN ← tL + ta(s);
8  when receive star−message(t)
9    if t ≠ tN then
10     error
11   y ← λ(s)
12   send y−message(y,t) to parent
13   s ← δint(s)
14   tL ← t;
15   tN ← tL + ta(s)
16 when receive x−message(x ∈ X,t)
17   if (tL ≤ t and t ≤ tN) == false then
18     error
19   s ← δext(s,t − tL,x)
20   tL ← t;
21   tN ← tL + ta(s);
```

Listing 3.2: DEVS-Koordinator nach [131]

```
1  Variables:
2    parent // parent coordinator
3    tL // time of last event
4    tN // time of next event
5  when receive init−message(t)
6    for each d ∈ D do
7      send init−message(t) to child d
8    tL ← max{tLi : d ∈ D}
9    tN ← min{tNi : d ∈ D}
10 when receive star−message(t)
11   if t ≠ tN then
12     error
13   d* ← Select({d ∈ D : tNi = tN})
14   send star−message(t) to d*
15   tL ← max{tLi : d ∈ D}
16   tN ← min{tNi : d ∈ D}
17 when receive x−message(x ∈ X,t)
18   if (tL ≤ t and t ≤ tN) == false then
19     error
20   for each x ∈ Zi,d|i = N do
21     send x−message(x,t) to child d
22   tL ← max{tLi : d ∈ D}
23   tN ← min{tNi : d ∈ D}
24 when receive y−message(y ∈ Y,t)
25   for each y ∈ Zi,d|i,d ≠ N do
26     send x−message(y,t) to child d
27   for each y ∈ Zi,d|d = N do
28     send y−message(y,t) to parent
29   tL ← max{tLi : d ∈ D}
30   tN ← min{tNi : d ∈ D}
```

3.2.2 Parallel-DEVS-Formalismus

Der PDEVS Formalismus nach Chow [19] ist eine Weiterentwicklung des Classic-DEVS-Formalismus. PDEVS führt neue Mechanismen zur Verarbeitung zeitgleicher Ereignisse ein. Grundsätzlich definiert DEVS zwei Arten von Ereignissen, externe und interne Ereignisse, die unabhängig voneinander auftreten können. Der Zeitpunkt für jedes interne Ereignis und damit verbunden der Zeitpunkt für das Senden eines Ausgangsereignisses wird von jedem Atomic-DEVS selbst bestimmt. Entsprechend den Kopplungsbeziehungen wird ein Ausgangsereignis zu einem Eingangsereignis eines anderen Atomic-DEVS, welches das externe Ereignis verarbeiten muss. Folglich kann dies bei einem Atomic-DEVS zum zeitgleichen Auftreten von internen und externen Ereignissen führen. Der PDEVS-Ansatz löst diesen Konflikt intern im Atomic-PDEVS. Hierfür wird die Modellspezifikation um eine neue Dynamikfunktion erweitert, welche beim parallelen Auftreten von externen und internen Ereignissen ausgeführt wird. Der Konflikt wird somit ausschließlich auf

Atomic-PDEVS-Ebene gelöst. Allgemein muss der PDEVS-Ansatz gemäß Zeigler [131] drei wesentliche Eigenschaften erfüllen, welche in Tabelle 3.5 zusammengefasst sind:

Tabelle 3.5: Wesentliche Eigenschaften des PDEVS-Ansatzes

Wesentliche Eigenschaften
Alle Atomic-PDEVS mit ein und demselben internen Ereigniszeitpunkt erzeugen ihr jeweiliges Ausgangsereignis und verarbeiten zum selben Zeitpunkt ihre Dynamikfunktion.
Jedes erzeugte Ausgangsereignis wird entsprechend den Kopplungsbeziehungen zum Eingangsereignis anderer PDEVS. Es können zeitgleich mehrere externe Ereignisse auftreten. Dazu verwalten Simulatoren und Koordinatoren intern einen sogenannten Ereignis-Bag.
Für das zeitgleiche Auftreten von externen und internen Ereignissen wird im Atomic-PDEVS eine zusätzliche Zustandsüberführungsfunktion definiert.

Ein Atomic-PDEVS ist wie Talbelle 3.6 definiert.

Tabelle 3.6: Atomic-Parallel-DEVS (PDEVS)

$\mathbf{PDEVS} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$		
X	$\mathrm{X} = \{(\mathrm{p}, \mathrm{v}) \mid \mathrm{p} \in \mathrm{IPorts}, \mathrm{v} \in \mathrm{X_p}\}$	Menge aller Eingangsereignisse
S		Menge der Zustandswerte
Y	$\mathrm{Y} = \{(\mathrm{p}, \mathrm{v}) \mid \mathrm{p} \in \mathrm{OPorts}, \mathrm{v} \in \mathrm{Y_p}\}$	Menge aller Ausgangsereignisse
$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}} : \mathrm{S} \rightarrow \mathrm{S}$	interne Zustandsüberführungsfunktion
$\delta_{\mathbf{ext}}$	$\delta_{\mathrm{ext}} : \mathrm{Q} \times \mathrm{X} \rightarrow \mathrm{S}$	externe Zustandsüberführungsfunktion
	$\mathrm{Q} = \{(\mathrm{s}, \mathrm{e}) \mid \mathrm{s} \in \mathrm{S}, 0 \leq \mathrm{e} \leq \mathrm{ta(s)}\}$	Menge totaler Zustände
	e (elapsed time)	verstrichene Zeit seit letztem Ereignis
$\delta_{\mathbf{con}}$	$\delta_{\mathrm{con}} : \mathrm{Q} \times \mathrm{X} \rightarrow \mathrm{S}$	Zustandsüberführungsfunktion bei zeitgleichen Ereignissen (Confluent-Transition-Function)
λ	$\lambda : \mathrm{S} \rightarrow \mathrm{Y}$	Ausgabefunktion
ta	$\mathrm{ta} : \mathrm{S} \rightarrow \mathbb{R}^{+}_{0,\infty}$	Zeitfortschrittsfunktion

Abbildung 3.3 zeigt schematisch das dynamische Verhalten eines Atomic-PDEVS beim zeitgleichen Eintreten von externen und internen Ereignissen. Hierbei bestimmt die konfluente Zustandsüberführungsfunktion $\delta_{\mathrm{con}}(\mathrm{s}, \mathrm{e}, \mathrm{x})$ aus dem aktuellen Zustand $\mathrm{s} \in \mathrm{S}$, der verstrichenen Zeit e und vorliegenden Eingangsereignissen $\mathrm{x} \in \mathrm{X}$ den Folgezustand s‘.

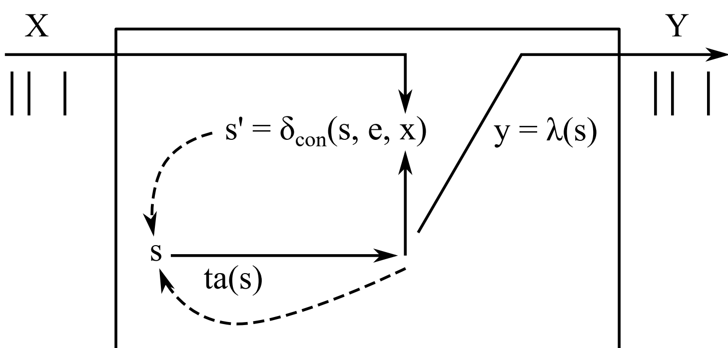


Abbildung 3.3: Dynamik eines Atomic-Parallel-DEVS bei zeitgleichem internen und externen Ereignis nach Zeigler [131]

Tabelle 3.8: Liste DEVS-basierter Simulatoren (unvollständig)

Umgebung	**Ausprägung**	**Sprache**	**Autoren**
PowerDEVS	Classic-DEVS	C++	Bergero und Kofman [9]
DEVS-Suite	Parallel-DEVS	Java	Kim, Sarjoughian und Elamvazhuthi [52]
CoSMoS	Parallel-DEVS	Java	Sarjoughian und Elamvazhuthi [94]
CD++ Builder	Classic-DEVS	C++	Wainer [122]
SimStudio	Classic-DEVS	Java	Traore [117]
VLE	Parallel-DEVS	C++	Quesnel et al. [89]
DesignDEVS	Classic-DEVS	Lua	Goldstein, Breslav und Khan [40]
MATLAB/DEVS	Parallel-DEVS	MATLAB	Deatcu, Schwatinski und Pawletta [28]
DEVSimPy	Classic-DEVS	Python	Capocchi et al. [14]

Da die Auflösung von Konflikten aufgrund zeitgleicher Ereignisse im PDEVS-Formalismus auf atomarer Ebene erfolgt, wird keine SELECT-Funktion mehr benötigt. Daraus folgen modifizierte Abarbeitungsalgorithmen für den Koordinator und Simulator. Die grundlegende Simulationsausführung entspricht aber dem Classic-DEVS-Ansatz. Ein Coupled-PDEVS ist wie wie in Tabelle 3.7 definiert.

Tabelle 3.7: Coupled-Parallel-DEVS (PDEVN)

$\mathbf{PDEVN} = (\mathbf{X}, \mathbf{Y}, \mathbf{D}, \{\mathbf{M_d}\}, \{\mathbf{I_d}\}, \{\mathbf{Z_{i,d}}\})$			
$\mathbf{X}$	$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$		Menge aller Eingangswerte
$\mathbf{Y}$	$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$		Menge aller Ausgangswerte
$\mathbf{D}$			Indexmenge der Subkomponenten
$\mathbf{M_d}$	Modellbeschreibung der Komponenten für $d \in D$ mit		
	$M_d = \{X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta\}$		als ein atomic PDEVS oder PDEVN
$\mathbf{I_d}$	ist die Menge aller Einwirkungen auf $d \in D$ mit $I_d \subseteq D \cup \{N\}, d \notin I_d$		
$\mathbf{Z_{i,d}}$	Menge der $i-to-d$ Kopplungensbeziehungen mit $i \in I_d$, für die gilt:		
	$Z_{i,d} : X \to X_d$,	wenn $i = N$	externe Eingangskopplung
	$Z_{i,d} : Y_i \to Y$,	wenn $d = N$	externe Ausgangskopplung
	$Z_{i,d} : Y_i \to X_d$,	wenn $d \neq N$ und $i \neq N$	interne Kopplung

Aufgrund wohldefinierter Simulationsalgorithmen und vieler Erweiterungsmöglichkeiten wird (P)DEVS in vielen Anwendungsfeldern genutzt. In Kang, Seo und Kim [50] wird beispielsweise ein komplexes Anwendungsszenario aus dem militärischen Bereich vorgestellt. Hierbei wird eine Vielzahl miteinander vernetzter Cyber-Physical-Systems (CPS) modelliert und anschließend simuliert. Die Simulationsergebnisse bilden die Grundlage zur Bewertung möglicher Strategien. Wie in Tabelle 3.8 exemplarisch gezeigt, existieren eine Vielzahl von Entwicklungsumgebungen für (P)DEVS.

Häufig besteht der Wunsch die Modellspezifikation eines atomaren DEVS-Modells graphisch zu beschreiben. In Kang, Seo und Kim [50] wird die DEVS-Diagramm-Notation verwendet. Nachfolgend wird diese vorgestellt und um neue Beschreibungsmittel erweitert. Die eingeführten Erweiterungen ergaben sich aus den Kontext dieser Arbeit. Das Ziel ist eine kompakte und übersichtliche Notation zur Dynamikbeschreibung atomarer (P)DEVS-Modelle. Die rein textuell-mengentheoretische Notation wird bei umfangreichen Modellen

schnell unübersichtlich.

3.2.3 DEVS-Diagramm-Notation und eingeführte Erweiterungen

Ein DEVS-Diagramm nach Song [111] ermöglicht die grafische Darstellung der Dynamik eines Atomic-DEVS-Modells. Das DEVS-Diagramm kann zur Spezifikation der Modelldynamik von Atomic-Classic-DEVS-Modellen und eingeschränkt für Atomic-PDEVS-Modellen genutzt werden. Die Notation sieht kein Beschreibungsmittel für die konfluente-Zustandsüberführung vor. Ein Atomic-(P)DEVS-Modell wird als ein Rechteck symbolisiert, wie in Abbildung 3.4 schematisch gezeigt.

Im oberen Bereich des Rechtecks wird der Name des atomaren Modells festgelegt. Am linken und rechten Rand des Rechtecks können Eingangs- bzw. Ausgangsports dargestellt werden. Ein Port ist durch einen Namen und den Wertebereich seiner Ereignisse charakterisiert. Die untere linke Ecke ist durch ein weiteres Rechteck vom Rest der Box abgegrenzt. Sie dient der Notation der Zustandsmenge, sowie des Initialzustandes. Die Notation folgt in der Regel dem Schema *Zustandsname:{Wertemenge}=Anfangszustand*.

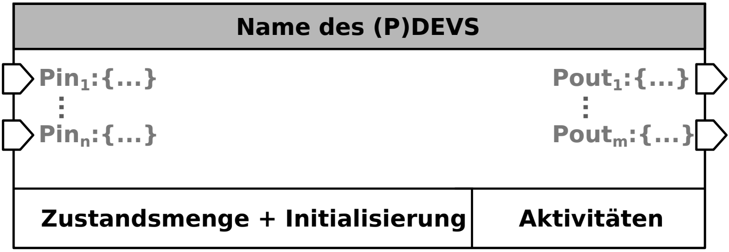


Abbildung 3.4: Schematische Darstellung eines Atomic-(P)DEVS mittels DEVS-Diagramm nach Song [111] einschließlich eingeführter Erweiterung *Aktivitäten*

Die Spezifikation der Dynamik erfolgt durch eine graphische Repräsentation der Hauptzustände. Hauptzustände entsprechen in der Regel der Zustandsmenge *Phase = {Active,Passive,...}*. Wie in Tabelle 3.9 gezeigt wird, werden Hauptzustände als Boxen mit abgerundeten Ecken symbolisiert. Zusätzlich zum Namen des Zustandes wird innerhalb der Box der Zeitpunkt des nächsten internen Ereignisses durch ein vorangestelltes @-Zeichen notiert. Dieses entspricht dem Ergebnis der Zeitfortschrittsfunktion ta(s).

Tabelle 3.9: DEVS-Diagramm-Notation und eingeführte Erweiterungen

Element	Beschreibung
Pin_n:{...}	**Eingangsport** mit Menge der Eingangsereignisse am Port `Pin`
Pout_m:{...}	**Ausgangsport** mit Menge der Ausgangsereignisse am Port `Pout`
State @t_N	**Zustandsdarstellung** Zustandsdarstellung und Zeitpunkt des nächsten internen Ereignis Abbildung der Zeitfortschrittsfunktion: $ta(s) \rightarrow t_N$
Pin?in	**Externe Zustandsüberführung** entspricht der $\delta_{ext}(s, e, x)$-Funktion wenn am Port `Pin` das Ereignis `in` auftritt
Pout!out	**Ausgabefunktion und interne Zustandsüberführung** Spezifikation der $\lambda(s)$-Funktion und $\delta_{int}(s)$-Funktion möglich
@[Bedingung]	**Transitionsbedingung** die Ausführung einer Transition kann an Bedingungen geknüpft sein (anwendbar auf externe oder interne Zustandsüberführung)
/{Aktion}	**Transitionsaktion** definiert Zustandsänderungen bei Ausführung einer Transition (anwendbar auf externe oder interne Zustandsüberführung)

Element	Erweiterungen zur DEVS-Diagramm-Notation
1 / 2	**Prioritäten** werden mehrere gleichartige Tansitionen spezifiziert, erfolgt ihre Abarbeitung in der durch die Nummer festgelegten Reihenfolge
1 / 2	**Conditon-Junktion** Spezifiaktion von Fallunterscheidungen und Zusammenfassung von Transitionen
Aktivitäten	**Aktivitäten** Liste der Wechselwirkungen mit Hard- und Softwarekomponenten (Darstellung ähnlich der Zustandsmenge)
#Kommentar	**Kommentar** Beschreibung der Intention (keine Auswirkung auf Modelldynamik)

Die durch Boxen dargestellten Hauptzustände können durch Kanten, welche Transitionen darstellen, miteinander verbunden werden. Die Darstellung ist ähnlich zu Zustandsautomaten (z.B. Statecharts). Jedoch werden zwei Arten von Transitionen unterschieden. Durch eine gestrichelte Linie wird eine interne Zustandsüberführungen und durch eine durchgezogene Linie eine Zustandsüberführung aufgrund eines externen Ereignisses an einem der Eingangsports, symbolisiert.

Die Beschriftung von Transitionen sollte nicht mit der Beschriftung von Transitionen in Zustandsautomaten verwechselt werden. Die Syntax beider Spezifikationen unterscheidet

sich voneinander. Eine interne Zustandsüberführung erfolgt immer auf Basis einer zuvor definierten Zeit t_N, welche durch ein vorangestelltes @-Zeichen innerhalb eines Hauptzustandes notiert ist. Ist diese vergangen, so werden die internen Transitionen auf ihre Gültigkeit geprüft und es werden gegebenenfalls ein Ausgangsereignis (Pin!out) erzeugt sowie der Zustandsübergang ausgeführt. Eine durchgezogene Kante repräsentiert eine externe Zustandsüberführung und ist mit den externen Ereignis (Pin?in) beschriftet. Zusätzlich können interne und externe Zuständüberführungen durch boolsche Transitionsbedingungen @[...] und Transitionsaktionen /{...} beschriftet werden.

Ein DEVS-Diagramm ermöglicht eine grafische Notation der Modelldynamik eines Atomic-DEVS-Modells. Jedoch führt eine komplexe Modelldynamik häufig zu einem komplexen Diagramm. Um die Darstellung der Diagramme zu vereinfachen, wurden Beschreibungsmittel aus der Notaion *heutiger* Statecharts übernommen. Hierbei können Transitionen mit *Prioritäten* beschriftet werden. Diese verbessern die Spezifikation einer deterministischen Abarbeitung eines durch DEVS-Diagramm beschriebenen Modells. Weiterhin wurde das Beschreibungsmittel Condition-Junktion eingeführt. Hierbei können größtenteils gleichartig beschrifte Transitionen zu einer Transition zusammengefasst werden. Außerdem wird durch das Beschreibungsmittel eine neue Möglichkeit zur Definition einer Fallunterscheidung geschaffen. Die neuen grafischen Beschreibungsmittel verbessern die Lesbarkeit der Notation von DEVS-Diagrammen.

Abbildung 3.5 zeigt ein Beispiel zur Spezifikation eines Atomic-Classic-DEVS, Prozessor (PROC), mittels DEVS-Diagramm. Das Modell verfügt über einen Eingangsport *Pin* und einen Ausgangsport *Pout*. Initialisiert wird das Modell mit dem Hauptzustand *Passive*. Bei einem externen Ereignis am Port *Pin* ändert sich der Hauptzustand in *Active*. Hierbei wird zugleich der Zustand σ auf den Wert t_{job} gesetzt. Hierdurch wird ein internes Ereignis zum Zeitpunkt $t + t_{job}$, mit $t =$ aktuelle Simulationszeit und job $=$ Jobbedienzeit, eingeplant. Bei Eintreten des internen Ereignisses nach σ-Zeiteinheiten, wird ein Ausgangsereignis am Port *Pout* mit dem Wert *done* erzeugt und durch Ausnutzung der internen Zustandsüberführungsfunktion erfolgt eine Transition in den Hauptzustand *Passive*. Das Modell befindet sich wieder im Iniatialzustand und ein neuer Zyklus kann beginnen.

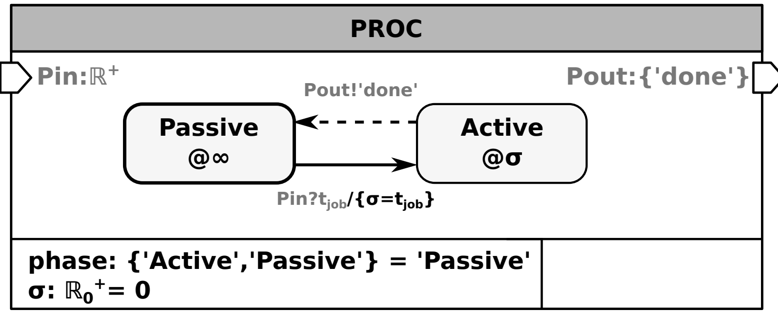


Abbildung 3.5: Spezifikation eines Atomic-Classic-DEVS namens PROC mittels DEVS-Diagramm

Zum Vergleich zur Darstellung des PROC mittels DEVS-Diagramm wird in Tabelle 3.10 die Mengennotation des gleichen Atomic-Classic-DEVS dargestellt.

Tabelle 3.10: PDEVS-Mengennotation eines PROC

$\mathbf{PROC} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \lambda, \mathbf{ta})$	
X	$X = \{(\mathrm{JOB}, t_{job}) \mid \mathrm{JOB} \in \mathrm{IPorts}, t_{job} \in \mathbb{R}^+_{0,\infty}\}$
S	$S = \{(\mathrm{phase}, \sigma) \mid \mathrm{phase} \in \{\text{‘Active‘}, \text{‘Passive‘}\}, \sigma \in \mathbb{R}^+_{0,\infty}\}$
	$s_{initial} = (\text{‘Passive‘}, \infty)$
Y	$Y = \{(\mathrm{DONE}, \text{‘done‘}) \mid \mathrm{DONE} \in \mathrm{OPorts}\}$
$\delta_{\mathbf{int}}$	$\delta_{int}((\mathrm{phase}, \sigma)) := (\text{‘Passive‘}, \infty)$
$\delta_{\mathbf{ext}}$	$\delta_{ext}((\mathrm{phase}, \sigma), e, (\mathrm{JOB}, t_{job})) := (\text{‘Active‘}, t_{job})$
λ	$\lambda((\mathrm{phase}, \sigma)) := (\mathrm{DONE}, \text{‘done‘})$
ta	$\mathrm{ta}((\mathrm{phase}, \sigma)) := \sigma$

Die weiterhin eingeführte Erweiterung *Aktivitäten* wird im nächsten Abschnitt benutzt und erläutert.

3.3 Echtzeit- und Prozessanbindung

Der DEVS Formalismus mit seinen zahlreichen Erweiterungen wird hauptsächlich im Bereich der Simulationstechnik verwendet. Mit der RT-DEVS-Erweiterung wurde ein spezieller Ansatz für die Echtzeitsimulation und für die Anbindung an reale Prozesse entwickelt. Daneben existieren auch Ansätze, die den Classic-DEVS beziehungsweise den Parallel-DEVS-Formalismus um Echtzeitfähigkeit und Schnittstellen zur Prozessanbindung erweitern. Nachfolgend werden wesentliche aus der Literatur bekannte Ansätze vorgestellt und bewertet.

3.3.1 Real-Time-DEVS-Formalismus

Grundlagen des Formalismus: Der RT-DEVS Formalismus nach Zeigler et al. [131] ist eine Erweiterung des Classic-DEVS-Formalismus. Er unterstützt die Ausführung von hierarchischen DEVS-Modellen in einer Echtzeitumgebung und berücksichtigt darüber hinaus die Interaktion mit externen Softwaresystemen und physikalischen Umgebungen, welche allgemein als externe Komponenten oder Prozesse bezeichnet werden.

In Zusammenhang mit RT-DEVS wird häufig der Begriff *Aktivität* verwendet, welcher eine zielgerichtete Wechselwirkung mit externen Komponenten bezeichnet. Allgemein hat eine Aktivität einen Startzeitpunkt und einen Endzeitpunkt, sowie eine zumeist nicht deterministische Ausführungszeit.

Modellspezifikation: Im Vergleich mit Classic-DEVS bleibt die Definition der Coupled-DEVS, genannt coupled-Real-Time-DEVS (RT-DEVN), nahezu unverändert. Wie auch bei PDEVN entfällt die Select-Funktion, vgl. Tabelle 3.11. Beim Auftreten gleichzeitiger Ereignisse wird immer ein Ereignis priorisiert verarbeitet. Eine Spezifikation, welches Ereignis priorisiert wird, ist seitens des Anwenders nicht möglich. Ein Coupled-Real-Time-DEVS ist wie in Tabelle 3.12 definiert.

Tabelle 3.11: Coupled-Real-Time-DEVS, auch genannt Real-Time-DEVS-Network (RT-DEVN)

$\mathbf{RT-DEVN = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})}$	
$\mathbf{X, Y, D, I_d, Z_{i,d}}$	analog Classic-DEVS
$\mathbf{M_d}$	ein RT-DEVS

Wie in Hong et al. [48] gezeigt wird, unterscheidet sich die Modellspezifikation eines atomaren RT-DEVS Modells nicht grundlegend von der Spezifikation eines Atomic-Classic-DEVS-Modells. Die Definition eines Atomic-Real-Time-DEVS zeigt Tabelle 3.12. Ergänzend zur Spezifikation von Atomic-Classic-DEVS-Modellen wird eine Aktivitätsfunktion ψ eingeführt, welche den aktuellen Zustand $s \in S$ auf eine Aktivität $a \in A$ abbildet. Im RT-DEVS-Formalismus sind Aktivitäten ausführbare Aktionen zur Wechselwirkung mit Hard- und Softwarekomponenten. Jeder Aktivität wird ein Zeitintervall, bestehend aus einer unteren und oberen Zeitgrenze, zugeordnet. Die Zuordnung wird mittels der neu eingeführten Zeitforschrittsfunktion ti realisiert. Damit die Einhaltung des Zeitintervalls überwacht werden kann, erfolgt beim RT-DEVS-Formalismus die Zeitfortschaltung auf Basis physikalischer Zeit und in Echtzeit.

Tabelle 3.12: Atomic-Real-Time-DEVS (RT-DEVS)

$\mathbf{RT-DEVS = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, ti, \psi, A)}$		
$\mathbf{X, S, Y, \delta_{int}, \lambda, ta}$		analog atomic-Classic-DEVS
$\boldsymbol{\delta_{ext}}$	$\delta_{ext} : Q \times X \rightarrow S$	externe Zustandsüberführungsfunktion
	$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ti(s)\}$	Menge totaler Zustände
	e (elapsed time)	Zeit seit letztem Ereignis
ti	$ti : S \rightarrow \mathbb{R}^+_{0,\infty} \times \mathbb{R}^+_{0,\infty}$	Zeitfortschrittsfunktion
ψ	$\psi : S \rightarrow A$	Aktivitätsfunktion
A		Menge der ausführbaren Aktivitäten

Operationelle Semantik: Das dynamische Verhalten eines RT-DEVS ist in Abbildung 3.6 dargestellt. Jedem Zustand $s \in S$ ist eine Aktivität $a \in A$ zugeordnet, welche in Echtzeit ausgeführt wird. Weiterhin gilt, dass eine Aktivität a weder Ereignisse empfangen, Ereignisse versenden oder Zustände verändern darf. Durch die Beendigung der aktuellen Aktivität a im Zeitintervall $ti(s) = [t(a)|_{min}, t(a)|_{max}]$ wird ein internes Ereignis im Atomic-RT-DEVS ausgelöst. Dabei wird allgemein angenommen, dass sämtliche Aktivitäten $a \in A$ nichtdeterministische Ausführungszeiten besitzen. Beim Eintreten eines internen Ereignisses wird zunächst die Ausgabefunktion $\lambda(s)$ ausgeführt und auf Grundlage des Zustandes $s \in S$ ein Ausgangsereignis $y \in Y$ erzeugt. Danach wird die interne Zustandsüberführungsfunktion $\delta_{int}(s)$ ausgelöst, welche auf Grundlage des inneren Zustandes $s \in S$ den nachfolgenden Zustand s‘ ermittelt. Anschließend wird für den berechneten Folgezustand $s‘ \in S$ die zugehörige Aktivität $a = \psi(s‘)$ angestoßen und das neue Zeitintervall ti(s‘) berechnet. Sobald ein externes Ereignis $x \in X$ eintritt, wird die aktuelle Aktivität a des Atomic RT-DEVS abgebrochen. Im Anschluss daran werden der Folgezustand $s‘ = \delta_{ext}(s, e, x)$ bestimmt und die zugehörige Aktivität $a‘ = \psi(s‘)$ angestoßen, sowie das neue Zeitintervall ti(s‘) bestimmt.

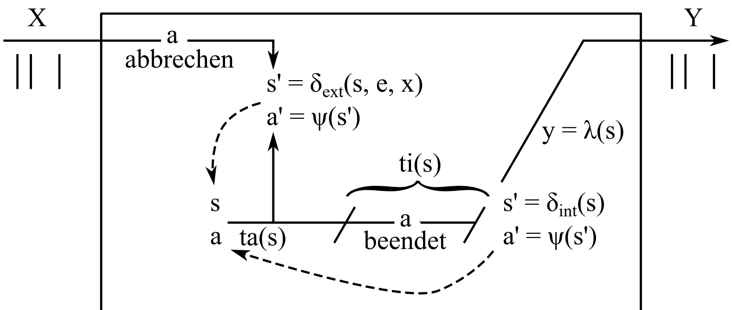


Abbildung 3.6: Dynamik eines Atomic-Real-Time-DEVS nach Zeigler [131]

Die Ausführung der Modellspezifikation entspricht grundlegend dem Classic-DEVS-Formalismus. Jedem Atomic-DEVS-Modell wird ein Simulator und jedem Coupled-DEVS-Modell ein Koordinator zugeordnet. Aus Effizienzgründen wird in der Regel ein abgeflachtes Modell erzeugt. Bei einem abgeflachten Modell werden die gekoppelten Modelle der Zwischenhierarchien aufgelöst, so dass eine Transformation in ein gekoppeltes Modell erfolgt, welches alle atomaren Modelle enthält. Die Simulation folgt nicht einer virtuellen Simulationszeit, sondern der Realzeit. Die Listings 3.3 bis 3.5 zeigen die wesentlichen Unterschiede der Abarbeitung bei RT-DEVS gegenüber Classic-DEVS.

Listing 3.3 zeigt die Behandlung eines internen Ereignisses durch den RT-DEVS-Simulator nach Hong et al. [48]. Dieser führt die interne Ereignisbehandlung (when receive star-message) nur aus, falls eine Aktivität a $\in$ A erfolgreich und ohne Unterbrechung durch ein externes Ereignis ausgeführt wurde. Zuerst prüft der Simulator die Einhaltung des zur Aktivität gehörenden Zeitintervalls. Eine Verletzung des Zeitintervalls führt zu einer Fehlermeldung. Wird das spezifizierte Zeitintervall eingehalten, werden die Dynamikfunktionen des Atomic-RT-DEVS-Modells in der Sequenz $[\lambda, \delta_{\text{int}}, \text{ta}, \text{ti}, \psi]$ ausgeführt. Die λ-Funktion kann zum Generieren von Ausgangsereignissen genutzt werden, welche den Kopplungsrelationen entsprechend zu Eingangsereignissen (external-events) anderer Atomic-RT-DEVS-Modelle werden. Nach dem Aufruf der λ-Funktion erfolgt die Aktualisierung des aktuellen Zustandes s $\in$ S durch die interne Zutandsüberführungsfinktion δ_{int} und das Einplanen des nächsten internen Ereigniszeitpunkts auf Basis der Zeitfortschrittsfunktion ta(s). Im Anschluss hieran wird die Ausführung der zum neuen Zustand gehörenden Aktivität durch die Bestimmung des zulässigen Zeitintervalls auf Basis von ti(s) eingeleitet und die neue Aktivität a‘ $\in$ A durch die Aktivitätsfunktion ψ(s) $\rightarrow$ a‘ gestartet. Um eine mögliche Diskrepanz zwischen der Simulationszeit und der physikalischer Zeit zu verringern, versucht der Simulator interne Ereignisse so genau wie möglich zum anversierten Zeitpunkt auszuführen und mögliche Abweichungen zu minimieren.

Listing 3.4 zeigt die Behandlung eines externen Ereignisses (when receive x-message) durch den Simulator eines Atomic-RT-DEVS-Modells nach Hong et al. [48]. Wie im vorherigen Listing 3.3, wird zuerst die Gültigkeit des Ereignisses auf Basis des Zeitintervalls geprüft. Eine Nichteinhaltung führt zu einer Fehlermeldung. Anderenfalls wird ein neuer Zustand durch das zum Simulator gehörende Atomic-RT-DEVS-Modell durch Ausführung der externen Zustandsüberführungsfunktion δ_{ext} bestimmt. Analog der Star-Message aus Listing 3.3 folgt die Einplanung des nächsten internen Ereigniszeitpunkts, die Bestimmung des Zeitintervalls der neuen Aktivität, sowie das Anstoßen der neuen Aktivität.

Listing 3.5 zeigt den Algorithmus der Simulationsschleife eines RT-DEVS-Simulators nach Hong et al. [48]. Zuerst findet die Initialisierung des zum Simulator zugeordneten Atomic-RT-DEVS-Modells statt. Hierbei wird zuerst der Zeitpunkt des nächsten internen Ereignisses durch die Zeitfortschrittsfunktion ta bestimmt. Anschließend wird das zulässige Zeitintervall durch die Zeitintervallfunktion ti ermittelt, gefolgt von der Bestimmung der aktuellen Aktivität $a \in A$ durch die Aktivitätsfunktion ψ.

Nachdem die Initialisierung abgeschlossen wurde, wird die Simulationsschleife betreten. Zuerst wird die Ausführung der zuvor ermittelten Aktivität angestoßen. Die weitere Ausführung der Simulationsalgorithmen wird anschließend blockiert bis ein *Signal* detektiert wird. Dieses kann entweder durch das erfolgreiche Beenden der Aktivität $a \in A$ erzeugt werden oder durch die Abarbeitung der Star-Message eines anderen Simulators, wenn dieser ein internes Ereignis verarbeitet. Zeile 5 in Listing 3.3 zeigt den Vorgang des Signalisierens. Dieser Vorgang geht mit dem Versenden eines Ausgangsereignisses einher. Dieses wird durch die Kopplungsbeziehungen zu einem Eingangsereignis eines anderen Atomic-RT-DEVS-Modells.

Um die Ursache des Signals zu ermitteln und eine entsprechende Signalbehandlung auszuführen, erfolgt zunächst eine Fallunterscheidung. Wenn die Ursache des Signal ein externes Ereignis ist, wird die Ausführung des *when receive x-message* Algorithmus in Listing 3.4 gestartet. Hierbei wird eine neue Aktivität $a \in A$ bestimmt. Die alte Aktivität $a \in A$ wird anschließend abgebrochen. Wenn keine Nachricht empfangen wurde, so ist die Ursache des Signals das Terminieren der aktuellen Aktivität $a \in A$ und es erfolgt der Aufruf des *when receive star-message* Algorithmus in Listing 3.3. Auch hierbei wird eine neue Aktivität $a‘ \in A$ bestimmt. In beiden Fällen wird diese zur aktuellen Aktivität ($a \leftarrow a‘$) bevor ein neuer Durchlauf der Simulationsschleife erfolgt.

Listing 3.3: star-Message des RT-DEVS-Simulators nach [48]

```
1  when receive star−message(Time t)
2  if ti_N|_min ≤ t ≤ ti_N|_max then
3     y ← λ(s)
4     send (y,t) to target executives
5     signal them
6     s ← δ_int(s)
7     try: compensate the timing error
        |t − t_N|
8     t_L ← t
9     t_N ← t_L + ta(s)
10    ti_N ← [t_L + ti(s)_min, t_L + ti(s)_max]
11    a‘ ← ψ(s)
12 else
13    error
```

Listing 3.4: x-Message des RT-DEVS-Simulator nach [48]

```
1  when receive x−message(Time t)
2  if t_L ≤ t ≤ ti_N|_max then
3     e ← t − t_L
4     s ← δ_ext(s,e,x)
5     t_L ← t
6     t_N ← t_L + ta(s)
7     ti_N ← [t_L + ti(s)|_min, t_L + ti(s)|_max]
8     a‘ ← ψ(s)
9  else
10    error
```

Listing 3.5: Algorithmus eines RT-DEVS-Simulators nach [48]

```
1  initialize:
2  t_N ← ta(s)
3  ti_N ← ti(s)
```

```
4   a ← ψ(s)
5   while(true)
6     invoke an activity a
7     wait for a signal
8     if a message is received then
9         when receive x−message(Time t)
10      cancel the activity a
11    else if the aktivity a is terminated then
12        when receive star−message(Time t)
13    a ← a‘
```

Anwendung in der Steuerungsentwicklung: Der RT-DEVS-Formalismus wurde entwickelt, um eine Ausführung von DEVS-basierten Modellen in Echtzeit zu unterstützen und Interaktionen mit einer Prozessumgebung zu ermöglichen. Die Zielstellung von RT-DEVS ist der Einsatz von DEVS im Rahmen der Steuerungsentwicklung und des operativen Betriebs. Hier stellt sich die Frage der Durchgängigkeit von Modellen und Werkzeugen von der Entwurfsphase bis zum operativen Betrieb. In der Entwurfsphase können Modelle mit Classic-DEVS entwickelt und mittels Simulation getestet werden. In der Automationsphase müssen die Modelle für den operativen Betrieb an RT-DEVS angepaßt werden. Hierfür sind die Modellspezifikationen der Atomic-DEVS-Modelle, um die zwei neuen Dynamikfunktionen $\psi(s)$ und $ti(s)$ zu ergänzen. Diese sind kein Bestandteil der Spezifikation von Atomic-Classic-DEVS Modellen und werden folglich durch einen Classic-DEVS-Simulator ignoriert. Die Abarbeitung der erweiterten Modellspezifikation bedingt einen Wechsel vom Classic-DEVS-Simulator zum RT-DEVS-Simulator. Tabelle 3.13 stellt die Spezifikation von Classic-DEVS und RT-DEVS einander gegenüber.

Tabelle 3.13: Vergleich der Spezifikation von Classic-DEVS-Modellen nach Zeigler et al. [131] mit RT-DEVS-Modellen nach Hong et al. [48]

atomic	**Classic-DEVS**	**RT-DEVS**
$X, S, Y, \delta_{int}, \lambda, ta$	identisch	identisch
δ_{ext}	$Q \times X \rightarrow S,$	$Q \times X \rightarrow S,$
	$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$	$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ti(s)\vert_{max}\}$
ti	nicht vorhanden	$S \rightarrow \mathbb{R}^+_{0,\infty} \times \mathbb{R}^+_{0,\infty}$
ψ	nicht vorhanden	$S \rightarrow A$
A	nicht vorhanden	$A = \{a \mid t(a) \in \mathbb{R}^+_{0,\infty}, a \notin \{X!, Y!, S =\}\}$
coupled	**Classic-DEVS**	**RT-DEVS**
$D, M_d, I_d, Z_{i,d}$	identisch	identisch
Select	$2^M - \phi \rightarrow M$	nicht vorhanden

In Cho [17] wird ein Framework zur Spezifikation und Ausführung von RT-DEVS-Modellen vorgestellt. Die RT-DEVS-Modelle sollen mit der Umgebung, gebildet aus Hard- und Softwarekomponenten, sowie menschlichen Interakteuren wechselwirken. Hierfür wird die Umsetzung einer RT-DEVS Infrastruktur in einer Softwareumgebung vorgestellt. Eine weitere Umsetzung des RT-DEVS-Formalismus wird in Zeigler [130] präsentiert. Der Einsatz von RT-DEVS im Kontext einer durchgängigen Steuerungsentwicklung wird in Cho [18] gezeigt. Wie in Abbildung 3.7 dargestellt, wird ein in der Entwurfsphase entwickeltes Modell schrittweise weiterentwickelt, um im operativen Betrieb als Steuerungssoftware

zu fungieren. Weiterhin wird Corba zur verteilten Ausführung der Simulation sowie der operativen Steuerung auf mehreren Rechnern genutzt.

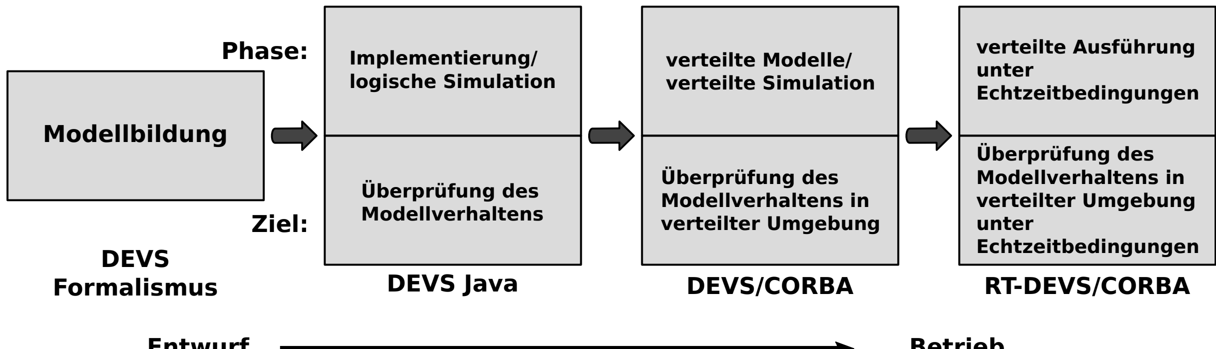
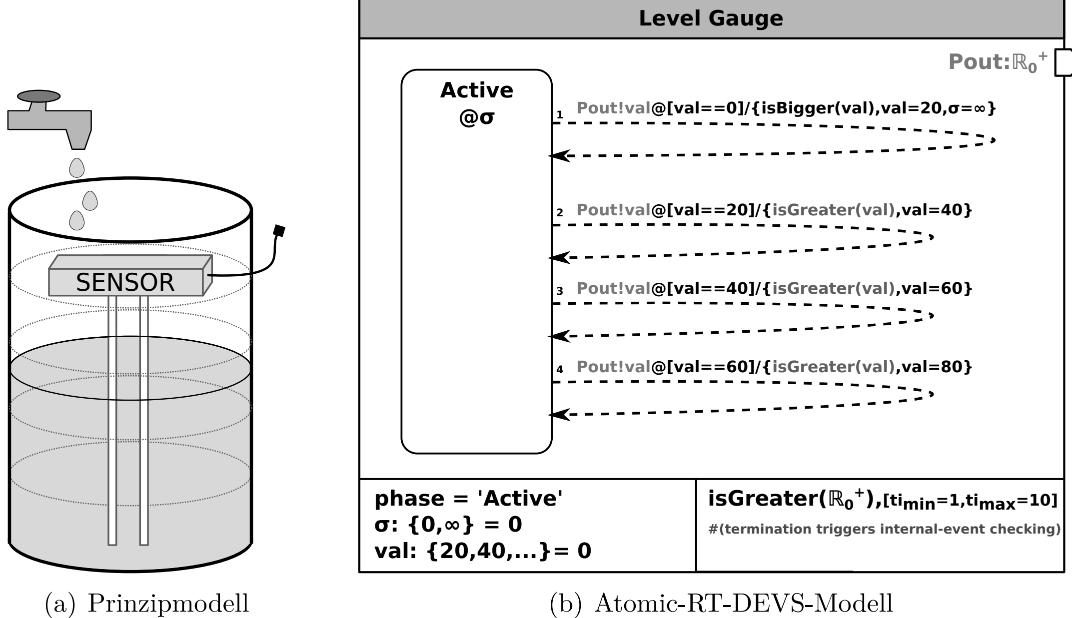


Abbildung 3.7: Vorgehensmodell der RT-DEVS/Corba-Umgebung nach Cho [18]

Wie die analysierten Arbeiten zeigen, eignet sich der RT-DEVS-Formalismus, um mit einer Prozessumgebung zu interagieren. Jedoch ist die Interaktion bestimmten Einschränkungen unterlegen. Laut der Definition des RT-DEVS-Formalismus, darf eine Aktivität a keine Ereignisse empfangen und versenden oder Zustände verändern.
Diese Einschränkungen ermöglichen es, dass eine Classic-DEVS-Modellspezifikation entwickelt, simulativ gestestet und zu einer RT-DEVS-Spezifikation mit Prozessanbindung weiterentwickelt werden kann. Die Weiterentwicklung zu RT-DEVS impliziert allerdings einen Wechsel der Abarbeitungsalgorithmen, dass heißt man benötigt eine RT-DEVS-Umgebung.

In Song [110] wird die Eignung des RT-DEVS Ansatzes für sicherheitskritische Anwendungen untersucht. Am Beispiel einer Bahnanlage wird der Entwicklungsprozess einer entsprechenden Steuerung auf Basis des RT-DEVS-Formalismus gezeigt. Die Hardwareinteraktion beschränkt sich auf die Integration binärer Sensoren. Diese detektieren das Auftreten sicherheitskritischer Prozesszustände.

Das eine Aktivität den Zustand des Modells nicht verändern darf und keine Ereignisse an andere Modelllkomponenten verschicken darf, schränkt die Anwendbarkeit des RT-DEVS-Formalimus ein, beziehungsweise führt zu *umständlichen* Modellspezifikationen. Zur Verdeutlichung dieser These wird in Abbildung 3.8 das Beispiel eines Füllstandssensors betrachtet. Dazu sind wesentliche Teile der Modellspezifikation mittels DEVS-Diagramm gezeigt.



(a) Prinzipmodell (b) Atomic-RT-DEVS-Modell

Abbildung 3.8: Problematische Umsetzung eines Füllstandssensors mittels RT-DEVS-Formalismus

Um den Sensorwert zu ermitteln, muss eine Aktivität definiert werden. Die Aktivität dient als Bindeglied zwischen dem realen Sensor und dem Modell. Nach den Algorithmen des RT-DEVS-Simulators kann eine Aktivität nur durch ein Ereignis signalisieren, dass sie abgeschlossen wurde. Ein Ereigniswert (Sensorwert) kann nicht zurückgegeben werden. Demzufolge muss der Wertebereich des Sensors auf Modellebene diskretisiert werden. Beim Beispiel erfolgt die Diskretisierung mittels der Zustandsvariablen val. Dem Hauptzustand Active ist die Aktivität *isGreater(val)* zugeordnet. Diese beendet sich und löst ein internes Ereignis aus, wenn der Sensorwert größer als der aktuell übergebene Schwellwert in val ist. Es wird ein Ausgangsereignis (Pout!val) generiert und eine interne Zustandsüberführung ausgelöst, bei welcher val aktualisiert wird. Anschließend wird die Aktivität *isGreater(val)* neu gestartet. Die Ausgangsereignisse des atomaren Modells repräsentieren lediglich Schwellwerte gemäß der vorgenommenen Diskretisierung.

3.3.2 Action-Level-Real-Time-DEVS

Grundlagen des Formalismus: In Sarjoughian und Gholami [95] wird der ALRT-DEVS Ansatz vorgestellt, der grundlegend auf den Ideen des RT-DEVS-Formalismus basiert aber auch Erweiterungen des Parallel-DEVS-Formalismus nutzt.

Modellspezifikation: Dementsprechend ähnelt die Spezifikation eines Atomic-ALRT-DEVS, der Spezifikation eines Atomic-RT-DEVS, wie in Tabelle 3.14 gezeigt ist. Die Spezifikation der Coupled-ALRT-DEVS entspricht der Spezifikation von PDEVN aus Unterabschnitt 3.2.2.

Tabelle 3.14: Atomic-Action-Level-Real-Time-DEVS nach Sarjoughian und Gholami [95]

$\mathbf{ALRT-DEVS} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \mathbf{\Omega}, \mathbf{\Gamma}, \lambda, \mathbf{ti}, \psi, \mathbf{A})$		
$\mathbf{X}, \mathbf{Y}$		analog atomic-PDEVS
S	$S = L \times Se$	Zustandsmenge
Ω	$\Omega = \{\delta_{int,h}\}, h \in \mathbb{N}$ $\delta_{int,h} : L \rightarrow L'$, $L = \{l_h\}, l_h = (q_h, \delta t_h), l_h \rightarrow a \in A, s_h \in S$, $q_h \subseteq s_h, \sum_{h=1}^{n} \delta t_h \leq \Delta t, \delta t_h \in ti(s)$	endliche Menge von δ_{int} Funktionen
Γ	$\Gamma = \{\delta_{ext,g}\}, g \in \mathbb{N}$ $\delta_{ext,g} : (L, e, x) \rightarrow L'$, $0 \leq e \leq \infty, x \in X$	endliche Menge von δ_{ext} Funktionen
λ	$\lambda : (S, A) \rightarrow Y$	Ausgabefunktion
ti	$ti : S \rightarrow \mathbb{R}^{+}_{[0,\infty)} \times \mathbb{R}^{+}_{(0,\infty]}$	
ψ	$\psi : L \rightarrow A$, $L = \{l_g\}, l_g = (q_g, \delta t_g), l_g \rightarrow a \in A, s_g \in S$, $q_g \subseteq s_g, \sum_{g=1}^{n} \delta t_g \leq \Delta t, \delta t_g \in ti(s)$	Aktivitätsfunktion
$\mathbf{A}$		Menge von ausführbaren Aktivitäten

Bei ALRT-DEVS ist die Zustandsmenge S durch das Kreuzprodukt der Menge der primären Zustände, genannt Locations L, und der Menge sekundärer Zustände Se definiert. Die Menge L wird zumeist durch das Kreuzprodukt der DEVS-spezifischen Zustandsvariablen phase und σ gebildet. Die Menge Se ist durch das Kreuzprodukt einer beliebigen Anzahl weiterer Zustandsvariablen $(v_1 \times v_2 \times ... \times v_n)$ definiert. Wie bei RT-DEVS wird eine Prozessanbindung über die Spezifikation von Aktivitäten, in der Originalliteratur *Actions* genannt, mit einem zulässigen Zeitintervall realisiert. Im Gegensatz zu RT-DEVS können Teilmengen von A definiert werden, sogenannte *ActivitySets*, die einem Zustand $l \in L$ zugeordnet werden. Weiterhin kann nicht nur eine externe beziehungsweise interne Zustandsüberführungsfunktion spezifiert werden, sondern jeweils eine Menge beider Funktionstypen (Ω, Γ). Der Grund dafür ist vermutlich eine verbesserte Strukturierung. Die Spezifikation der Ausgabefunktion λ beinhaltet, dass ein Ausgangsereignis $y \in Y$ basierend auf dem aktuellen Zustand $s \in S$ und der aktuell beendeten Aktivität $a \in A$ generiert werden kann. Im Gegensatz zu RT-DEVS kann die momentane Aktivität Einfluss auf den Wert des Ausgangsereignisses nehmen.

Operationelle Semantik: Wie auch bei RT-DEVS führt der ALRT-DEVS-Simulator ein Modell in Echtzeit basierend auf der physikalischen Zeit aus. Jedoch unterscheiden sich die Simulatoralgorithmen voneinander. Abbildung 3.9 zeigt den grundlegenden Simulationsalgorithmus von ALRT-DEVS. Bei der Initialisierung werden $ti(s)|_{min}, ti(s)|_{max}$ der aktuellen Aktivität $a \in A$ durch den Simulator bestimmt und die Aktivität gestartet (Abb.3.9, 1). Tritt ein externes Ereignis ein (Abb.3.9, 2), wird die Ausführung der Aktivität unterbrochen und eine externe Zustandsüberführung mit $\delta_{ext,g}$ ausgeführt (Abb.3.9, 3). Wurde die Location $l \in L$ nicht verändert, wird die Ausführung der aktuellen Aktivität fortgesetzt (Abb.3.9, 1). Im anderen Fall wird das zu $l' \in L$ zugehörige *ActivitySet* ausgewählt (Abb.3.9, 5) und die erste Aktivität des Sets als aktuelle Aktivität gesetzt(Abb.3.9, 6), das zugehörige Zeitintervall $ti(s)|_{min}$, $ti(s)|_{max}$ bestimmt (Abb.3.9, 7) und die Aktivität zur Ausführung gebracht (Abb.3.9, 1).

Wird eine Aktivität im zulässigen Zeitintervall beendet (Abb.3.9, 8), wird mit λ ein Ausgangsereignis generiert (Abb.3.9, 9) und geprüft, ob es die letze Aktivität des aktuellen *ActivitySets* war (Abb.3.9, 10). Wenn nein, wird die nächste Aktivität zur Ausführung gebracht (Abb.3.9, 11). Im anderen Fall wird eine internen Zustandsüberführung mit $\delta_{\mathrm{int,h}}$ ausgeführt (Abb.3.9, 12) und das zur Location $\mathrm{l}' \in \mathrm{L}$ zugehörige *ActivitySet* ausgewählt (Abb.3.9, 5) und die erste Aktivität des Sets als aktuelle Aktivität gesetzt (Abb.3.9, 6) und ein neuer Zyklus gemäß der vorangegangenen Beschreibung gestartet.

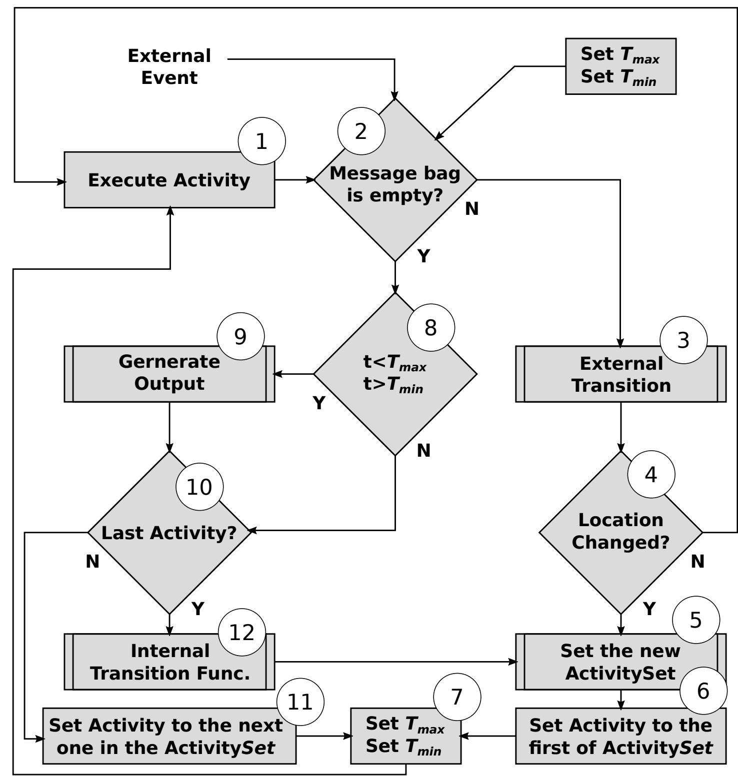


Abbildung 3.9: Simulationsalgorithmus des ALRT-DEVS-Ansatzes von Sarjoughian und Gholami [95]

Zusammenfassend folgt bezüglich der Ausführung von Aktivitäten:

- Jede Aktivität $\mathrm{a} \in \mathrm{A}$ eines ActivitySet wird ohne Unterbrechung ausgeführt, solange ihr jeweiliges Zeitfenster nicht verlassen wird und ein externes Ereignis nicht zum Wechsel der Location $\mathrm{l} \in \mathrm{L}$ führt.
- Wird eine Aktivität $\mathrm{a} \in \mathrm{A}$ nicht innerhalb ihres definierten Zeitfensters $\mathrm{ti(s)}|_{\mathrm{min}} \leq \mathrm{t} \leq \mathrm{ti(s)}_{\mathrm{max}}$ abgeschlossen, wird sie zwangsweise beendet und falls vorhanden, die nachfolgende Aktivität $\mathrm{a}‘ \in \mathrm{A}$ gestartet. Nur die Aktivitäten, welche innerhalb ihres definierten Zeitfensters ausgeführt werden konnten, führen zum Aufruf der λ Funktion und zur Erzeugung eines Ausgangsereignisses $\mathrm{y} \in \mathrm{Y}$.
- Für den Fall, dass sich eine Aktivität zum selben Zeitpunkt beendet zu dem ein externes Ereignis eintritt, wird die Erzeugung eines Ausgangsereignisses durch die Aktivität standardmäßig priorisiert. Das externe Ereignis geht in diesem Fall verloren und wird nicht ausgewertet.

Anwendung in der Steuerungsentwicklung: In Sarjoughian und Gholami [95] wird

das Modell eines Netzwerk-Switches mit Hilfe von ALRT-DEVS modelliert und untersucht. Das Modell untersucht zwei mögliche Strategien zur Verteilung von Netzwerk-Paketen. Hierbei ermöglicht der ALRT-DEVS-Ansatz, dass ein SM in Echtzeit unter Einbeziehung simulierter und realer Prozesskomponenten ausgeführt werden kann. Die Nutzung der entwickelten Modelle als Steuerungssoftware für den operativen Betrieb nach dem RCP-Ansatz nach Abel [1] wird von den Autoren nicht vorgeschlagen, ist jedoch prinzipiell möglich. Wird ein SM in der Entwurfsphase zunächst mittels Classic-DEVS spezifiziert, ist kein direkter Übergang zum ALRT-DEVS-Formalismus möglich, da sich die Modellspezifikationen und Abarbeitungsalgorithmen unterscheiden. Die Zielsetzung des ALRT-DEVS-Formalismus ist vermutlich explizit auf die Umsetzung von Simulationen unter Einbeziehung realer Hardware ausgerichtet und nicht auf ein phasenorientiertes Vorgehensmodell zur Entwicklung von Steuerungen fokussiert.

3.3.3 Real-Time-Embedded-DEVS

Grundlagen des Formalismus: In Moallemi [73] wird der RTE-PDEVS Formalismus für die schrittweise simulationsgestützte Entwicklung von Hardwareanwendungen eingeführt. Hierbei verspricht Moallemi eine nahtlose Integration von Simulationsmodellen auf eingebetteten Hardwarekomponenten. Ein Simulationsmodell soll schrittweise von einem ersten Entwurfsmodell bis zum operativen Betrieb weiterentwickelt werden. Wie Moallemi anführt, lassen sich die meisten Software-Methoden nicht gut auf große Projekte skalieren. Das heißt, einzelne Komponenten sind schwer zu testen. Das Ziel seiner Arbeit besteht in der Entwicklung eines durchgängig anwendbaren Ansatzes zur Steuerungsentwicklung und entspricht demgemäß den Anforderungen des RCP.

Modellspezifikation: Ein erster Modellentwurf sowie erste Tests erfolgen auf Basis des PDEVS-Formalismus aus Unterabschnitt 3.2.2. Das entwickelte Simulationsmodell wird anschließend um ein Interface zur Ansteuerung von Hard- und Softwarekomponenten erweitert. Dieses wird durch eine veränderte Spezifikation des in der Hierarchie höchsten Coupled-RTE-PDEVS (RTE-PDEVN) erreicht. Die Spezifikation des RTE-PDEVN ist in Tabelle 3.15 gezeigt. Alle anderen Coupled-PDEVS bleiben unverändert. Das RTE-PDEVN hat die Aufgabe mit einem Echtzeit-Treiber-Modul (RT-DM) zu kommunizieren. Spezielle Schnittstellen koppeln das RTE-PDEVN bidirektional mit dem RT-DM. Somit kann ein Atomic-RTE-PDEVS über Ausgangsereignisse mit dem RTE-PDEVN und dieses mit dem RT-DM kommunizieren. Zusätzlich können Hardwarezustände über externe Ereignisse an das RTE-PDEVN und wiederrum an ein Atomic-RTE-PDEVS verschickt werden.

Tabelle 3.15: Coupled-RTE-PDEVS (RTE-PDEVN) nach Moallemi [74]

$\mathbf{RTE-PDEVN = (X, Y, IS, OS, Dx, DY, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})}$		
$\mathbf{X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}}$		analog PDEVN
IS	$IS = \{(is, iy, dl)\}$ mit	Menge der Hardwareeingangssignale mit Deadline
	$is \in$	Eingangssignale der Hardware
	$iy \in Y$	Ausgangsport welcher das Ergebnis des Eingangssignals erhält
	$dl \in \mathbb{R}^+_{0,\infty}$	Deadline des Eingangssignals
OS	$OS = \{(os, oy, pt)\}$ mit	Menge der Hardwareausgangssignale
	$os \in$	Ausgangssignale zur Hardware
	$oy \in Y$	Ausgangsport welcher das Signal erhält
	$pt \in \mathbb{R}^+_{0,\infty}$	Verarbeitungszeit vom Erhalt des Signals is
DX	$DX : IS \to Xv$	Hardwareeingangssignale zu Eingangsport-Wert (Xv)
DY	$DY : Yv \to OS$	Ausgangsport-Werte zu Hardwareausgangssignalen (Yv)
	$\forall\ iy = oy \to pt \leq dl$	Randbedingung

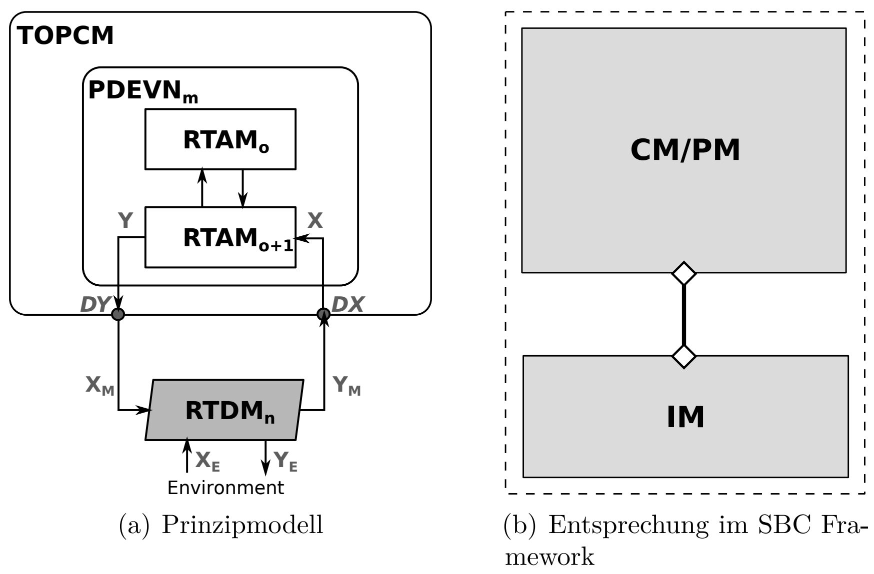
Die Spezifikation eines RT-DM ist in Tabelle 3.16 dargestellt. Das RT-DM verfügt über eine Menge von Ein- und Ausgangsports zur Kommunikation mit einem RTE-PDEVN. Externe Ereignisse werden durch eine Ereignistransformationsmethode auf Interaktionen mit der Hard- und Softwareumgebung abgebildet. Weiterhin wird eine Methode zur Abbildung relevanter Umgebungswerte auf Ereignisse spezifiziert.

Tabelle 3.16: Real-Time-DEVS-Treiber (RT-DM) nach Moallemi [74]

$\mathbf{RT-DM = (X, Y, T_{ME}, T_{EM})}$		
X	$X = X_M \cup X_E$	Menge aller Eingangsereignisse
	$X_M :$	Eingangsereignisse vom Modell
	$X_E :$	Eingangsereignisse der Umgebung
Y	$Y = Y_M \cup Y_E$	Menge aller Ausgangsereignisse
	$Y_M :$	Ausgangsereignisse zum Modell
	$Y_E :$	Ausgangsereignisse zur Umgebung
$\mathbf{T_{ME}}$	$T_{ME} : X_M \to Y_E$	Ereignisumformungsmethode (Modell zur Umgebung)
$\mathbf{T_{EM}}$	$T_{EM} : X_E \to Y_M$	Ereignisumformungsmethode (Umgebung zum Modell)

Operationelle Semantik: Im Unterschied zum PDEVS-Fomalismus arbeitet die Zeitfortschrittsfunktion bei RTE-PDEVS auf Basis physikalischer Zeit, welche in diesem Zusammenhang Wall-Clock-Time (WCT) genannt wird. Beim RTE-DEVS-Ansatz wird der abstrakte-DEVS-Simulator verändert. Ziel der Änderungen ist die Unterstützung von Hardwaresteuerungsmechanismen, welche die Integration von externen Hardwarekomponenten unterstützen. Da der RTE-PDEVS-Formalismus auf dem PDEVS-Formalismus aufsetzt, sind in den späteren Entwicklungsphasen nur minimale Änderungen der entwickelten Modellspezifikationen erforderlich. Nach Moallemi unterstüzt der RTE-PDEVS-Formalismus eine durchgängige Modellnutzung und Verifikation einmal entwickelter PDEVS-Modelle. Der Autor verspricht, dass bei Anwendung seines Ansatzes eine erhöhte Zuverlässigkeit und Übertragbarkeit besteht. Die konkreten Simulationsalgorithmen werden in [73] nicht gezeigt. Jedoch wird ein Verweis auf das downloadfähige DEVS-Toolkit E-CD++ gegeben, welches die in Moallemi disktutierten Anpassungen beinhaltet.

Anwendung in der Steuerungsentwicklung: Erste Untersuchungen wurden in Moallemi [74] am Beispiel der Ansteuerung eines einfachen LEGO-Roboterarms gezeigt. Ein Beispiel der Struktur eines RTE-DEVS Modells ist in Abbildung 3.10 (a) dargestellt. Eine Wechselwirkung des Modells mit der Hardware wird über das Real-Time-Driver-Model (RT-DM) erreicht. Dieses ist kein klassisches DEVS-Modell, wenngleich es auch über kompatible Ports zum Empfang und Versenden von Ereignissen durch das RTE-PDEVN (TOPCM) verfügt. Das TOPCM ist ein, um spezielle Schnittstellen erweitertes, Coupled-PDEVS (PDEVN) und kann weitere PDEVS-Modelle enthalten.



(a) Prinzipmodell

(b) Entsprechung im SBC Framework

Abbildung 3.10: Schematische Darstellung eines RTE-RDEVS-Modells und Einordnung in den SBC-Ansatz

Bei einem Vergleich mit dem SBC-Ansatz nach Abschnitt 2.3.3, fällt auf, dass das RT-DM dem IM entspricht. Das TOPCM kann als eine Mischung aus CM und PM interpretiert werden. Dieser Zusammenhang ist in Abbildung 3.10 (b) illustriert. Eine explizite Trennung von Prozess und Steuerung wird im Kontext des RTE-PDEVS-Formalismus nicht vorgeschlagen.

3.3.4 DEVS-Based Transparent M&S Framework

Grundlagen des Formalismus: Der TR-DEVS Formalismus nach Risco Martin [90] ist ein Ansatz, um ein Classic-DEVS um eine Prozessanbindung zu erweitern. Hierdurch wird eine durchgängige Weiterentwicklung eines in der Entwurfsphase entwickelten Modells unterstützt. Das zugrundeliegende Prinzip kann nach [90] auf den PDEVS-Formalismus übertragen werden.

Modellspezifikation: Bei TR-DEVS bleiben die Modellspezifikation der Atomic- und Coupled-Modelle im Vergleich zu Classic-DEVS unverändert. Die Anbindung notwendiger Hardware erfolgt über die Definition sogenannter JOBs. Ein Job abstrahiert eine Menge von Wechselwirkungen mit der Prozessumgebung, um ein gewünschtes Ziel zu erreichen. Die zugrundeliegende Idee beruht auf dem Konzept des virtuellen Dateisystems, wie es bei

UNIX-ähnlichen Systemen konsequent umgesetzt wird, *alles ist Datei.* Das grundlegende Prinzip ist in Abbildung 3.11 dargestellt.

Ein atomares DEVS kann bei Bedarf in ein sogenanntes DeviceFile (f) schreiben. Eine nebenläufige Routine der zugrundeliegenden Laufzeitumgebung überwacht diese Datei. Werden Änderungen detektiert, so wird ein zum *DeviceFile* gehörender *DeviceDriver* aktiv. Dieser setzt die gewünschte Interaktion mit der Hardware um. Gleichzeitig werden Änderungen des Zustandes der Hardware erfasst und in das *DeviceFile* zurückgeschrieben. Das Lesen und Schreiben des *DeviceFiles* erfolgt mit den Zustandsüberführungsfunktionen δ_{int} oder δ_{ext}. Zusätzlich kann ein lesender Zugriff auch innerhalb der Ausgabefunktion λ spezifiziert werden.

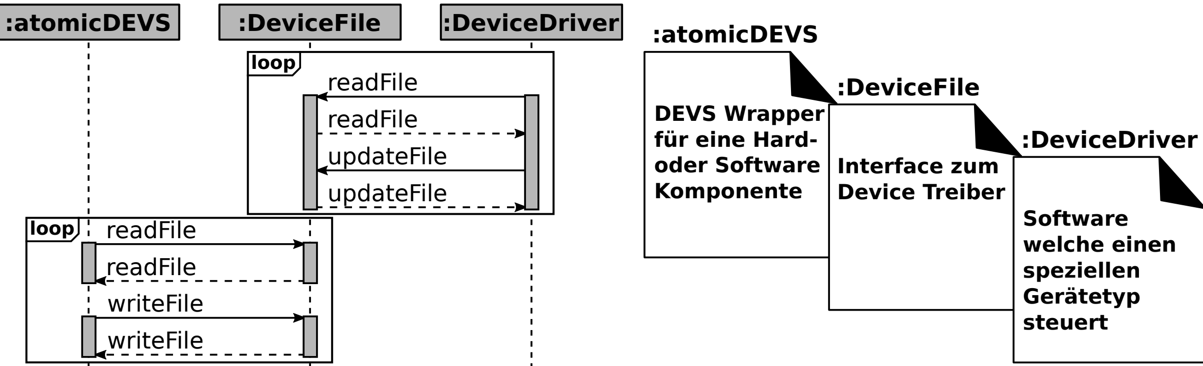


Abbildung 3.11: Schematische Darstellung elementarer Komponenten des Transparent-DEVS-Frameworks nach Risco Martin [90]

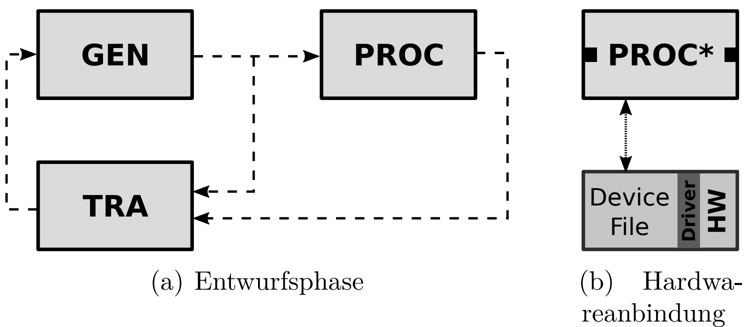
Um eine spontane Änderung einer Prozesskomponente zu detektieren, muss gepollt werden. Hierbei plant ein atomares Modell mit $\text{ta}(s) = 0$ ein internes Ereignis, damit infolgedessen seine interne Zustandsüberführungsfunktion δ_{int} ausgeführt wird. Bei jedem Aufruf prüft diese das DeviceFile auf Änderungen. Wird hierbei eine Änderung detektiert, so kann ein Ausgangsereignis erzeugt und gemäß den Kopplungsrelationen an andere Komponenten verschickt werden.

Operationelle Semantik: Im Gegensatz zu RT-DEVS, ALRT-DEVS oder RTE-DEVS wird für die Ausführung eines TR-DEVS-Modells kein neuer Simulator benötigt. Stattdessen kann ein Classic-DEVS oder entsprechend ein PDEVS-Simulator verwendet werden.

Anwendung in der Steuerungsentwicklung: Abbildung 3.12 (a) zeigt ein Entwurfsmodell einer einfachen Produktionsanlage mit einem Teilefluss. Diese wird durch drei Atomic-Classic-DEVS-Modelle abgebildet. Ein Generator (GEN) simuliert einen ankommenden Teilestrom nach einer mathematischen Verteilungsfunktion. Die ankommenden Teile werden durch einen Prozessor (PROC) bearbeitet. Der PROC kann in der Entwurfsphase als ein Server mit einer konstanten Bedienzeit modelliert werden. Nachdem ein Bauteil durch den PROC bearbeitet wurde, wird eine Nachricht über die erfolgreiche Fertigstellung eines Produkts an eine Komponente Transducer (TRA) verschickt. Der TRA überwacht den Prozess und kann die Produktion beim Erreichen einer vorgegebenen Zielgröße stoppen. Hierfür wird ein externes Ereignis vom TRA zum GEN geschickt.

Abbildung 3.12 (b) zeigt den Übergang eines Entwurfsmodells in die Automatisierungsphase. Zur Integration externer Hardware ist eine Modifikation der Komponente PROC

erforderlich. Tabelle 3.17 zeigt die notwendige Modifikation der Modellkomponente PROC. Die neue Komponente PROC* verfügt über eine Schnittstelle zu einem realen Prozess.



(a) Entwurfsphase (b) Hardwareanbindung

Abbildung 3.12: Modell einer Produktionsanlage bestehend aus Generator (GEN), Prozessor (PROC) und Transducer (TRA) nach Risco-Martin [90]

Über den SiL Ansatz nach Abschnitt 2.3.2 kann beispielsweise ein Roboter angesteuert werden. Hierfür muss ein entsprechendes DeviceFile und ein dazugehöriger DeviceTreiber definiert werden. Das Atomic-Classic-DEVS PROC* kann mithilfe seiner dynamikbeschreibenden Funktionen in das DeviceFile schreiben und hierdurch Steuerungsbefehle an das Robotersystem absetzen. Weiterhin wird das DeviceFile zyklisch gelesen, um den aktuellen Zustand des Robotersystems zu überwachen. Eine Definition eines zulässigen Ausführungsintervalls ist nicht möglich, da der zugrundeliegende Classic-DEVS-Simulator mit virtueller Zeitfortschaltung arbeitet. Eine Synchronisation zwischen virtueller (logischer) und physikalischer Zeit ist scheinbar nicht vorgesehen. Die Ausarbeitung in [90] erfolgt an Hand eines Demonstrationsbeispiels, das keine endgültige Beurteilung der vollständigen Funktionalität zulässt.

Tabelle 3.17: Atomic-Transparent-DEVS (TR-DEVS)

$\mathbf{TR-DEVS} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \lambda, \mathbf{ta})$		
X	$\mathrm{X} = \{(\mathrm{in}, \mathrm{j} \in \mathrm{J})\}$	J is a set of Jobs
S	$\mathrm{S} = (\sigma \times \mathrm{phase} \times \mathrm{J})$	$\mathrm{phase} \in \{\text{‘busy‘}, \text{‘passive‘}\}$
Y	$\mathrm{Y} = \{(\mathrm{out}, \mathrm{j} \in \mathrm{J})\}$	
$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}}(\sigma, \mathrm{phase}, \mathrm{j}) = (\infty, \text{‘passiv‘}, \emptyset)$	
	PROC	**PROC***
$\delta_{\mathbf{ext}}$	$\delta_{\mathrm{ext}}(\sigma, \mathrm{phase}, \mathrm{j}, \mathrm{e}, (\mathrm{in}, \mathrm{j_p}\text{‘})) = (\mathrm{j_p}\text{‘}, \text{‘busy‘}, \mathrm{j}\text{‘})$	$\delta_{\mathrm{ext}}(\sigma, \mathrm{phase}, \mathrm{j}, \mathrm{e}, (\mathrm{in}, \mathrm{j_p}\text{‘})) = (\mathrm{j_p}\text{‘}, \text{‘busy‘}, \mathrm{j} \leftarrow \mathrm{f})$
	$\delta_{\mathrm{ext}}(\sigma, \mathrm{phase}, \mathrm{j}, \mathrm{e}, (\mathrm{in}, \mathrm{j_p}\text{‘})) = (\sigma - \mathrm{e}, \text{‘busy‘}, \mathrm{j})$	$\delta_{\mathrm{ext}}(\sigma, \mathrm{phase}, \mathrm{j}, \mathrm{e}, (\mathrm{in}, \mathrm{j_p}\text{‘})) = (\sigma - \mathrm{e}, \text{‘busy‘}, \mathrm{j})$
λ	$\lambda(\sigma, \mathrm{phase}, \mathrm{j}) = \mathrm{j}$	$\lambda(\sigma \times \mathrm{phase} \times \mathrm{J}) = \mathrm{f} \leftarrow \mathrm{j}$
ta	$\mathrm{ta}(\sigma, \mathrm{phase}, \mathrm{j}) = \sigma$	$\mathrm{ta}(\sigma, \mathrm{phase}, \mathrm{j}) = \sigma$

3.3.5 PDEVS-RCP-Formalismus

Grundlagen des Formalismus: Der PDEVS-RCP-Formalismus ist eine Erweiterung des PDEVS-Formalismus und wird in Schwatinski [103] detailliert beschrieben. PDEVS-RCP erweitert PDEVS um die Möglichkeit der Interaktion mit externen Software- oder Hardwarekomponenten unter Echtzeitanforderungen.

Modellspezifikation: Die Definition der Atomic-PDEVS-RCP ist in Tabelle 3.18 dargestellt. Die Namensgebung PDEVS-RCP weist auf die enge Verwandtschaft zu PDEVS hin und auf die durchgängige Nutzbarkeit der Modelle gemäß dem Vorgehensmodell des RCP-Ansatzes nach Abel und Bollig [1].

Tabelle 3.18: Atomic PDEVS RCP

$\mathbf{PDEVS-RCP} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta}, \mathbf{A})$		
$\mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \mathbf{ta}$		analog atomic-PDEVS
$\mathbf{X}$	$X = X_{model} \cup X_{clock}$	Menge aller Eingangswerte
	$X_{model} = \{(p, v) \mid p \in IPorts, v \in X_p\}$	Menge der Eingangsereignisse des Modells
	$X_{clock} = \{(\text{‘clock‘}, v) \mid v \in \mathbb{R}^+\}$	Menge der Eingangsereignisse von einer Echtzeituhr (Real-Time-Clock, RTC)
$\mathbf{A}$		Menge der ausführbaren Aktivitäten
λ	$\lambda : S \to Y \times A$	kombinierte Ausgabe- und Aktivitätsfunktion

Die Definitionen von $Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, ta$ sind analog PDEVS. Die neu eingeführte Menge A spezifiziert die Interaktionen mit der Umgebung (Aktivitäten). Für jede Interaktion ist ein zulässiges Zeitinterval $[t_{min}\, t_{max}]$ anzugeben, welches die Echtzeitanforderungen festlegt. Die Menge der Eingangsereignisse setzt sich aus den Mengen X_{model} und X_{clock} zusammen. Dabei umfasst X_{model} die Eingangsereignisse gewöhnlicher PDEVS-Modelle sowie von der Umgebung. Die Menge X_{clock} mit $v \in R^+$ wird von einer Echtzeituhr generiert. Die kombinierte Ausgabe- und Aktivitätsfunktion λ definiert analog zu PDEVS die Berechnung von Ausgangsereignissen. Diese können an Modellkomponenten oder die Umgebung versendet werden. Weiterhin erfolgt durch die Ausgabefunktion λ die Bindung der aktuellen Aktivität $a \in A$ an den aktuellen Zustand $s \in S$. Analog dazu wird auch das zulässige Zeitintervall der aktuellen Aktivität a im aktuellen Zustand s abgebildet.

Operationelle Semantik: Das dynamische Verhalten eines Atomic-PDEVS-RCP-Modells zeigt Abbildung 3.13. Wie in in der Abbildung gezeigt, erfolgt die Verarbeitung externer Ereignisse durch die externe Zustandsüberführungsfunktion δ_{ext}. Die Auflösung von zeitgleichen internen und externen Ereignissen mittels der confluent-Funktion δ_{con} ist in der Abbildung nicht dargestellt, da sie identisch zu PDEVS ist. Wie in Schwatinski [103] dargestellt, legt diese in der Regel fest, ob für den aktuellen Zustandsübergang zuerst δ_{int} oder δ_{ext} auszuführen ist.

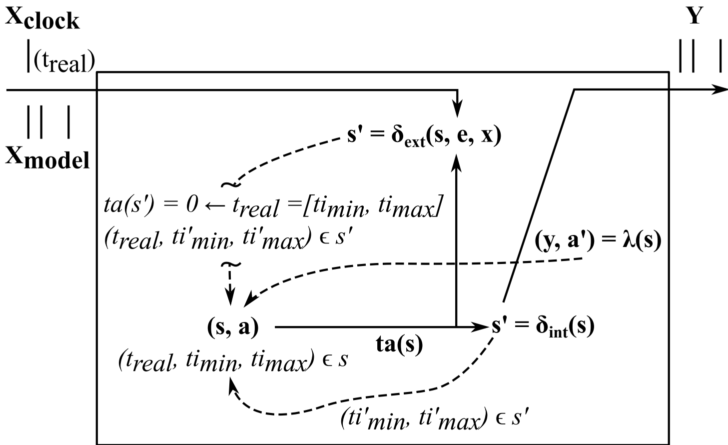


Abbildung 3.13: Dynamik eines Atomic-PDEVS-RCP basierend auf Schwatinski [103]

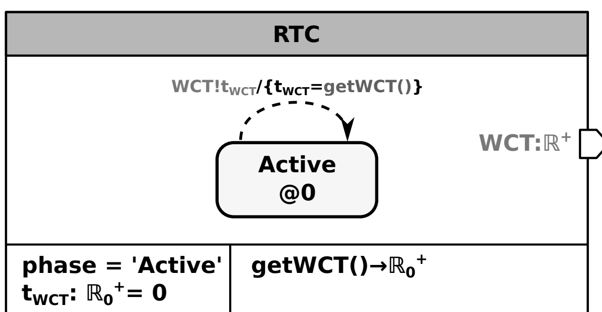


Abbildung 3.14: Atomic-PDEVS-RCP-Spezifikation einer Echtzeituhr basierend auf Schwatinski [103]

Anwendung in der Steuerungsentwicklung: Die Echtzeitsimulation eines Atomic-PDEVS-RCP basiert auf drei Voraussetzungen, welche in Tabelle 3.19 zusammengefasst sind.

Tabelle 3.19: Voraussetzungen der Echtzeitsimulation eines Atomic-PDEVS-RCP

Voraussetzungen des PDEVS-RCP-Fomalismus
Die minimalen und maximalen Ausführungszeiten der Aktivitäten müssen in der Zustandsmenge S des Atomic-PDEVS-RCP spezifiziert sein. Die Bestimmung der für einem Zustand gültigen Werte $t(s)_{min}$ und $t(s)_{max}$ erfolgt durch die Dynamikfunktionen.
Da die Ausführungsalgorithmen des PDEVS Formalismus, unverändert für die Echtzeitsimulation genutzt werden sollen, erfolgt die Ablaufsteuerung nach wie vor auf Basis der virtuellen Simulationszeit. Für atomare PDEVS-RCP Komponenten ist der virtuelle Zeitfortschritt ta(s) entweder 0 oder ∞.
Jedes Simulationsmodell, welches in Echtzeit simuliert wird, enthält genau ein Atomic-PDEVS Modell einer Echtzeituhr (RTC) gemäß dem DEVS-Diagramm in Abbildung 3.14. Diese sendet die physikalische Zeit als Ausgangsereignis an alle PDEVS-RCP Modelle, welche diese als externes Ereignis verarbeiten. Die Funktion *getWCT()* berechnet die *Wall-Clock-Time* unter Nutzung des zugrundeliegenden Betriebssystems

Da PDEVS-RCP unmittelbar auf PDEVS aufbaut, können PDEVS-RCP-Modelle direkt mit einer PDEVS-Simulationsumgebung ausgeführt werden, wobei diese aufgrund der eingeführten Erweiterungen auch als Echtzeitumgebung genutzt werden kann. Dadurch ist es möglich, PDEVS-RCP-Modelle schrittweise von der Entwurfsphase bis zum operativen Steuerungsbetrieb zu erweitern und mit einer einzigen Simulationsumgebung auszuführen. Damit erfüllt PDEVS-RCP die prinzipiellen Voraussetzungen zur softwaretechnischen Umsetzung des SBC-Frameworks.

3.4 Zusammenfassung

Heutige Steuerungsentwicklungen bedingen die Umsetzung einer Vielzahl von Anforderungen und erfordern zumeist ausführliche Tests. Hierbei erfolgt der Entwicklungsprozess häufig in Phasen. Jede Phase verfolgt separate Teilziele. Hierbei besteht der Wunsch, ein entwickeltes Modell über mehrere Phasen zu nutzen und weiter zu entwickeln. Im diesem Kapitel wurde der DEVS-Formalismus hinsichtlich des Einsatzes in der Steuerungsentwicklung analysiert. Als Modellierungs- und Simulationsformalismus findet er in der Entwurfsphase seine Anwendung. Beim Übergang in die Automatisierungsphase wird eine Echtzeit- und Prozessanbindung erforderlich. In diesem Zusammenhang wurden fünf aus der Literatur bekannte DEVS-Formalismen untersucht.

Der RT-, ALRT- und RTE-DEVS-Formalismus erfordern den Tausch des Simulators beim Übergang von der Entwurf- zur Automatisierungsphase. Weiterhin bestehen beim RT-DEVS-Formalismus Restriktionen, inwieweit sich externe Hardwarekomponenten steuern lassen. Dies erschwert die Weiternutzung eines zuvor entwickelten Simulationsmodells für die Steuerungsentwicklung. Beim TR-DEVS- sowie PDEVS-RCP-Formalismus erfolgt kein Wechsel des Simulationsalgorithmus. Die Integration externer Hardwarekomponenten wird auf Modellebene umgesetzt. Insbesondere der PDEVS-RCP-Formalismus ist restriktiv wie diese zu erfolgen hat. Eine Übernahme von Modellen der Entwurfsphase in die Automatisierungsphase ist mit entsprechenden Anpassungen auf Modellebene verbunden. Beim TR-DEVS-Formalismus erfolgt die Anbindung externer Hardwarekomponenten ohne wesentliche Restriktionen auf Ebene der Modellspezifikation. Sowohl beim PDEVS-RCP- als auch bei TR-DEVS-Formalismus müssen alle Modellkomponenten den Zustand externer Hardwarekomponenten durch Pollen abfragen. Dies führt zum Einplanen einer Vielzahl interner Ereignisse und somit zu Overhead. Weiterhin erfolgt beim TR-DEVS-Formalismus scheinbar keine Synchronisation zwischen virtueller (logischer) und physikalischer Zeit, zumindest wird diese in der Ausarbeitung in [90] nicht explizit erwähnt.

Zusammenfassend kann eingeschätzt werden, dass RTE-PDEVS, TR-DEVS und PDEVS-RCP eine Reihe konzeptioneller Gemeinsamkeiten besitzen. Gemäß der Literaturrecherche wurden TR-DEVS und PDEVS-RCP in etwa zeitgleich entwickelt. Aufbauend auf der Analyse in diesem Kapitel wird im nächsten Kaptitel ein neuer modifizierter PDEVS-RCP-Ansatz entwickelt, welcher insbesondere darauf abzielt den analysierten Overhead zu reduzieren.

4 Aufgabenorientierte Steuerungen für MRS mit SBC und DEVS

Basierend auf dem im Unterabschnitt 2.3.3 beschriebenen SBC Ansatz und den in Abschnitt 3.2 vorgestellten DEVS Formalismen, wird in diesem Kapitel ein neuer Ansatz zur durchgängigen Entwicklung von Multi-Robotersteuerungen vorgestellt. Hierfür werden zuerst die Probleme bestehender DEVS-Ansätze analysiert und anschließend ein modifizierter DEVS-Formalismus entwickelt. Danach wird an einem Fallbeispiel eine durchgehende Steuerungsentwicklung vom Entwurf bis zur Betriebsphase mit dem neuen Formalismus aufgezeigt, bevor explizit auf Multi-Robotersysteme (MRS) eingangen wird. Bei Betrachtung der MRS stehen insbesondere die Wechselwirkungen zwischen den Robotersystemen, nachfolgend Interaktionen genannt, im Fokus. Es wird gezielt die Abbildung von Interaktionen auf Aufgaben im Sinne einer aufgabenorientierten Steuerung (Task-Oriented-Control, TOC) untersucht. Wie in Abschnitt 2.4 gezeigt wurde, ist die Aufgabenorientierung eine etablierte Abstraktion in der Robotik, die Wiederverwendbarkeit und modular-hierachische Strukturierung unterstützt. Unter Ausnutzung dieser Eigenschaften der TOC werden für alle im Unterabschnitt 2.5 eingeführten Interaktionsklassen konzeptionelle Lösungsansätze entwickelt. Abschließend erfolgt eine Bewertung der neu entwickelten Methoden.

4.1 DEVS im Kontext der durchgängigen Steuerungsentwicklung

Wie in Abschnitt 2.3 diskutiert, basiert eine durchgängige Steuerungsentwicklung in der Regel auf einem Vorgehensmodell. In Maletzki [67] wird der SBC Ansatz für die durchgängige Steuerungsentwicklung nicht interagierender Single-Robotersysteme (SRS) verwendet. Die softwaretechnische Umsetzung erfolgte in der MATLAB-Umgebung. Die Arbeiten in Maletzki [67] und Li et al. [57] zeigen jedoch, dass die durchgängige Entwicklung ereignisorientierter Steuerungen mit den Standard-MATLAB-Werkzeugen (Simulink, Stateflow, SimEvents) aufgrund nicht theoriekonformer Implementierungen der Werkzeuge aufwendig ist. Li et al. sowie Maletzki empfehlen daher die Anwendung eines Formalismus und dessen konsequenter Umsetzung in Softwarewerkzeugen. Nachfolgend wird die Verwendung des DEVS-Formalismus im Kontext der durchgängigen Steuerungsentwicklung untersucht. Wie im Kapitel 3 gezeigt, basiert der DEVS-Formalismus auf einem systemtheoretischen Ansatz und es existieren bereits DEVS-Erweiterungen für die Steuerungsentwicklung.

4.1.1 Probleme bestehender DEVS-Ansätze

Das Ziel einer Entwurfsmethodik ist es, einen durchgängigen Entwicklungsprozess, wie in Abbildung 4.1 gezeigt, zu gewährleisten. Für die Durchgängigkeit ist es erforderlich, dass sich in der Entwurfsphase entwickelte Modelle phasenübergreifend weiterentwickeln lassen. Da in unterschiedlichen Phasen verschiedene Teilziele verfolgt werden, ist dies manchmal schwierig. Hilfreich ist hierbei die frühzeitige Betrachtung der Modellstrukturierung.

Das Ziel der Entwurfsphase ist die Erprobung und Bewertung möglicher Steuerungsstrategien. Hierfür wird der zu steuernde Prozess zumeist stark abstrahiert. Die Erprobung der unterschiedlichen Strategien erfolgt durch Simulationsexperimente. Wurde eine geeignete Strategie erarbeitet, so sollen die bereits entwickelten Modellkomponenten in der nächsten Phase, der Automatisierungsphase, wiederverwendbar sein. Dazu sollte frühzeitig, wie im Abschnitt 2.3.3 gezeigt, eine Separierung in ein Kontrollmodell (Control-Model, CM) und ein PM erfolgen.

In der Automatisierungsphase muss eine Schnittstelle zum zu steuernden Prozess entwickelt werden, welche in Abschnitt 2.3.3 als IM eingeführt wurde. Das Ergebnis der Automatisierungsphase sind verfeinerte Modellkomponenten mit höherer Komplexität. Die Erprobung erfolgt auch in dieser Phase per Simulation. Da bereits ein IM als Schnittstelle zum realen Prozess entwickelt wurde, können auch reale Steuerungserprobungen nach dem SiL Prinzip erfolgen. Auf diese Weise ist ein nahtloser Übergang in die Betriebsphase möglich, welcher gemäß Abschnitt 2.3.3 auch mittels Codegenerierung erfolgen kann.

Beim Auftreten von Fehlern sollte ein Vorgehensmodell die Wiederholung eines vorherigen Entwicklungsschrittes ermöglichen. Die Anforderungen der schrittweisen Verfeinerung von Modellen sowie Softwarekomponenten und die Wiederholung von Entwicklungsschritten kann durch Verwendung eines möglichst einheitlichen Formalismus über alle Entwicklungsphasen unterstützt werden. Wie in Abbildung 4.1 dargestellt, soll nachfolgend der DEVS-Formalismus diesbezüglich untersucht werden.

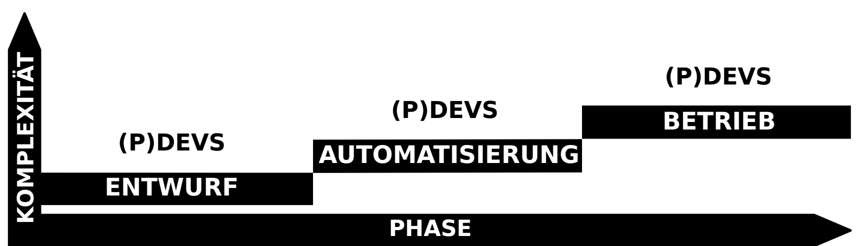


Abbildung 4.1: Gewünschte durchgängige Steuerungsentwicklung mit (P)DEVS

Der DEVS-Formalismus definiert nach Abschnitt 3.2 eine eindeutige Trennung von Modellbeschreibung und Abarbeitungsalgorithmen. Dadurch vereinfacht sich die Modularisierung, Strukturierung und Weiterentwicklung von Modellen. Die prinzipiellen Probleme beim Übergang von der Entwurfsphase in die Automatisierungsphase werden allein dadurch aber noch nicht gelöst. Die Modelle der Komponenten mit Prozessinteraktion müssen um Schnittstellen zur Kommunikation mit den Prozesskomponenten, dem IM, erweitert werden. Über das IM werden Aktorsignale an die realen Prozesskomponenten versendet und Sensorsignale empfangen. Wie in Derler et al. [30] und Deatcu et al. [29] dargestellt, entstehen durch die Prozessinteraktion neue Anforderungen. Einerseits muss die Abarbei-

tung des gesamten Modells echtzeitsynchronisiert erfolgen und andererseits ist die nicht zeitlose Ausführung von Aktionen zu berücksichtigen. So benötigt die Abarbeitung von Befehlen auf einer CPU sowie das Auslesen von Sensoren und die Ansteuerung von Aktorik reale Zeit. Die benötigte Zeit ist vorab nicht bekannt und unterliegt einer Vielzahl von Störungsgrößen. Im Extremfall können auf Grund von Übertragungsfehlern Anweisungen oder Rückmeldungen verloren gehen. In der Planungsphase werden derartige Einflüsse durch stochastische Größen abgebildet. Beim Übergang in die Automatisierungsphase muss eine Konkretisierung gemäß der realen Prozessinteraktionen erfolgen. Gemäß der Strukturierung nach dem SBC-Ansatz sind diese Störungen im IM zu modellieren.

Im Kapitel 3 wurden Erweiterungen der beiden originären DEVS-Formalismen Classic-DEVS und PDEVS, hinsichtlich Echtzeitfähigkeit und Schnittstellen zur Prozessanbindung analysiert sowie bezüglich der Anwendung im Rahmen einer durchgehenden Steuerungsentwicklung bewertet. Dabei zeigte sich, dass nur der TR-DEVS- und der PDEVS-RCP-Formalismus eine durchgehende Nutzung ohne Wechsel der Abarbeitungsalgorithmen unterstützen, da Schnittstellen zur Prozessanbindung auf Modellebene umgesetzt werden. Kritisiert wurde bei beiden Formalismen die Art und Weise der Interaktion mit externen Prozesskomponenten, die zu einen erheblichen Overhead führen kann. Beim TR-DEVS-Formalismus kommt hinzu, dass keine Synchronisation mit der Realzeit (WCT) erfolgt. Aus den genannten Gründen wird nachfolgend auf den PDEVS-RCP-Formalismus nach Schwatinski [103] aufgebaut. Mit der Weiterentwicklung sollen der bei PDEVS-RCP noch notwendige Anpassungsaufwand der Modelle beim Übergang in die Automatisierungsphase sowie der mögliche Overhead bei der Interaktion mit externen Prozessen verringert werden.

4.1.2 Entwicklung des PDEVS-RCP-V2-Formalismus

Um mit einer Prozessumgebung interagieren zu können, müssen die DEVS-Modelle beim Übergang in die Automatisierungsphase erweitert werden. Aufbauend auf der in Abschnitt 3.3 eingeführten Terminologie werden Prozessinteraktionen auf DEVS-Modellebene als Aktivitäten spezifiziert. Eine Aktivität ist beispielsweise die Ausführung eines Bewegungsbefehls durch einen Roboter. Im Kontext der Modellbildung und Simulation ist eine Aktivität durch ein Start- sowie Endereignis und die dazwischenliegende Zeitdauer charakterisiert. Die Ausführungszeit einer Aktivität ist für das Ablaufverhalten einer Steuerung wichtig. Basierend auf den Ausführungszeiten, können nach [103, 12] zum Beispiel fehlertolerante Steuerungen realisiert werden. Die Steuerung prüft, ob ein zuvor definiertes Zeitfenster seitens der Aktivität eingehalten wird und wenn nicht, wird eine Fehlerbehandlung eingeleitet. Weiterhin ist in der Steuerungstechnik oft der Verlauf einer Aktivität von Interesse, welcher mit Statusabfragen geprüft wird. Das Starten, Abfragen des Status und das Ende einer Aktivität sind dabei in der Regel mit Parameterübergaben verbunden.

PDEVS-RCP-V2: Analog zum PDEVS-RCP-Formalismus in Abschnitt 3.3.5 wird die Modellspezifikation so modifiziert, dass keine Anpassung der Ausführungsalgorithmen beim Übergang in die Automatisierungsphase erfolgen muss. Ausgehend von der Tatsache, dass die Kommunikationsschnittstelle von externen Prozesskomponenten in der Regel aus einer Menge von API-Methoden besteht, werden bei PDEVS-RCP-V2 Aktivitäten formal mit einer Menge von Function-Specified-System (FNSS) definiert. Nach Zeigler et al. [131] ist ein FNSS durch eine Eingangsmenge X, Ausgangsmenge Y und eine Ausgabefunktion λ

mit $\lambda : X \to Y$ charakterisiert. Gemäß dem dynamischen Verhalten eines FNSS können nur asynchron arbeitende API-Methoden als Ausgabefunktion eines FNSS definiert werden. Abbildung 4.2 zeigt schematisch die Vereinigung eines Atomic-PDEVS-Modells mit einer Menge von FNSS zu einem Atomic-PDEVS-RCP-V2. Prozessinteraktionen werden durch Ereignisse im PDEVS ausgelöst, welche die Ausführung von FNSS bewirken. Die Rückgabewerte der API-Methoden beeinflussen als Ausgangsereignisse Y der FNSS wiederum die Dynamik des PDEVS.

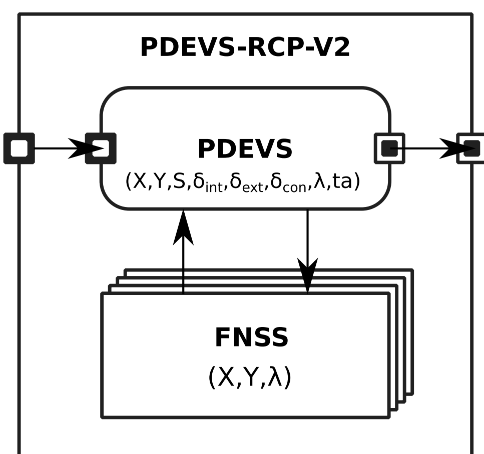


Abbildung 4.2: Kombination eines Atomic-PDEVS mit einer Menge FNSS zur Umsetzung von Prozessinteraktionen in PDEVS-RCP-V2

Tabelle 4.1 zeigt die Vereinigung der formalen Spezifikation von Atomic-PDEVS und einer Menge von FNSS zu Atomic-PDEVS-RCP-V2. Aus der Spezifikation ist die Verwandtschaft mit PDEVS-RCP nach Schwatinski [103] ersichtlich. Ein wesentlicher Unterschied zum originären PDEVS besteht in der Definition der Zustandsmenge S_{RCP}. Diese ist definiert als Vereinigung der Mengen S und A, wobei S der gewöhnlichen Zustandsmenge von PDEVS entspricht. Die Menge A spezifiziert die Aktivitäten. Jede Aktivität definiert eine Menge von FNSS, wobei die Ausgabefunktion λ jedes FNSS durch eine asynchron arbeitende Methode $m \in M$ definiert ist. Deshalb wird nachfolgend synonym der Begriff Methode verwendet. Die Aktivitäten, die als Teilmenge der neuen Zustandsmenge S_{RCP} definiert sind, beeinflussen mit den Ausgangsereignissen der FNSS unmittelbar die Zustandsmenge. FNSS werden auf Grund interner und externer Ereignisse aktiv und demzufolge in den Zustandsüberführungsfunktionen δ_{int}, δ_{ext} oder δ_{con} ausgeführt.

Wie aus der Spezifikation in Tabelle 4.1 ersichtlich, besitzt ein Atomic-PDEVS-RCP-V2 die gleichen Schnittstellen bezüglich der Abarbeitungsalgorithmen wie ein Atomic-PDEVS. Weiterhin besitzen beide eine identische operationelle Semantik, sodass ein PDEVS-RCP-V2 wie ein gewöhnliches PDEVS als Komponente eines PDEVN genutzt werden kann. Daraus folgt, dass ein beliebig modular-hierachisches PDEVN mit integrierten PDEVS-RCP-V2 mit den gewöhnlichen PDEVS-Abarbeitungsalgorithmen gemäß Abschnitt 3.2.2 ausgeführt werden kann. Dadurch kann eine durchgehende Weiterentwicklung von Modellen beginnend in der Entwurfsphase bis zum operativen Betrieb erfolgen.

Tabelle 4.1: Atomic-PDEVS-RCP-V2

$\mathbf{PDEVS-RCP-V2} = (\mathbf{X}, \mathbf{S_{RCP}}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$		
X	$X = \{(p,v) \vert p \in IPorts,\, v \in X_p\}$	Menge aller Eingangswerte
$\mathbf{S_{RCP}}$	$S_{RCP} = S \cup A$	Menge aller Zustände
	S	analog atomic-PDEVS
	$A = \{a_1, ..., a_i, ..., a_n \vert n \in \mathbb{N}\}$	Menge an Aktivitäten
	$a_i = \{m_1, ..., m_j, ..., m_o \vert$ $m_j : S_{RCP} \to S_{RCP}, o \in \mathbb{N}\}$	eine Aktivität als Menge von FNSS
Y	$Y = \{(p,v) \vert p \in OPorts,\, v \in Y_p\}$	Menge aller Ausgangswerte
$\delta_{\mathbf{int}}$	$\delta_{int} : S_{RCP} \to S_{RCP}$	interne Zustandsüberführungsfunktion
$\delta_{\mathbf{ext}}$	$\delta_{ext} : Q \times X \to S_{RCP}$	externe Zustandsüberführungsfunktion
	$Q = \{(s,e) \vert s \in S_{RCP},\, 0 \le e \le ta(s)\}$	Totaler Zustand
	e	vergangene Zeit seit letztem Ereignis
$\delta_{\mathbf{con}}$	$\delta_{con} : Q \times X \to S_{RCP}$	konfluente Zustandsüberführungsfunktion
λ	$\lambda : S_{RCP} \to Y$	Ausgabefunktion
ta	$ta : S_{RCP} \to \mathbb{R}^+_{0,\infty}$	Zeitfortschrittsfunktion

Echtzeitsynchronisation: Das, in der Entwurfs- und Automatisierungsphase entwickelte SM basiert auf einer virtuellen Zeitfortschaltung (Virtual-Time, VT) von einem Ereigniszeitpunkt zum nächsten. Beim Übergang in den operativen Betrieb muss die Abarbeitung synchron mit dem realen Zeitfortschritt (Wall-Clock-Time, WCT) erfolgen. Da die Abarbeitungsalgorithmen nicht modifiziert werden sollen, muss die Echtzeitsynchronisation auf Modellebene erfolgen. Dazu wird wie bei PDEVS-RCP [103] eine spezielle Modellkomponente Real-Time-Clock (RTC) eingeführt. Die neu eingeführte RTC basiert auf der PDEVS-RCP-V2-Spezifikation und behebt bestehende Probleme der alten PDEVS-RCP-basierten RTC. Die Unterschiede der beiden Ansätze werden im nächsten Abschnitt anhand eines Beispiels diskutiert.

Abbildung 4.3 und Tabelle 4.2 zeigen die Spezifikation der PDEVS-RCP-V2-basierten RTC in Mengennotation sowie als erweitertes DEVS-Diagramm. Die RTC synchronisiert die VT mit der WCT. Der Zustand $s \in S$ ist definiert als 4-Tupel $s = (phase, \sigma, t_{Last}, t_{WCT})$. Weiterhin ist eine Aktivität a mit der Methode getWCT() definiert. Diese implementiert die Prozessinteraktion mit der Laufzeitumgebung und liefert die WCT als Rückgabewert. Der Initialzustand s ist definiert mit (‘Active‘, 0, 0, 0). Zum Initialzeitpunkt $t = 0$ plant RTC ein internes Ereignis ein. Die zwei reflexiven Kanten (Transitionen) auf den Zustand ‘Active‘ spezifizieren eine Fallunterscheidung für δ_{int}.

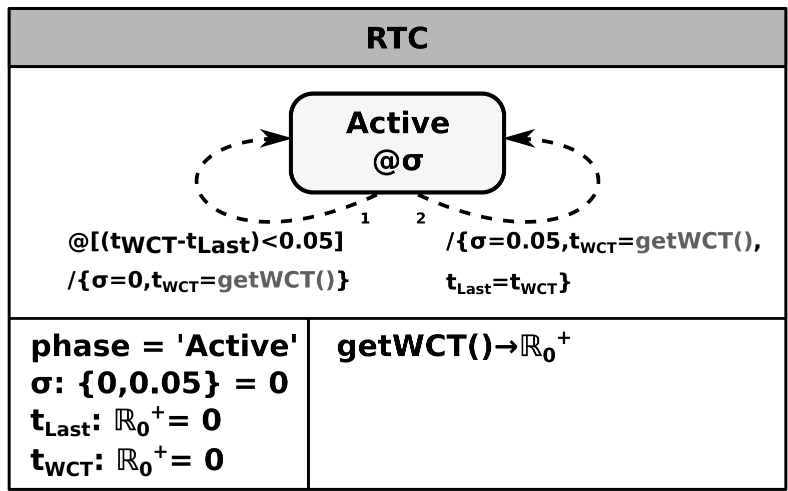


Abbildung 4.3: Spezifikation der Real-Time-Clock (RTC) nach PDEVS-RCP-V2

Tabelle 4.2: PDEVS-Mengennotation einer RTC

$\mathbf{RTC} = (\mathbf{X}, \mathbf{S_{RCP}}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$	
$\mathbf{X}$	$\mathrm{X} = \varnothing$
$\mathbf{S_{RCP}}$	$\mathrm{S_{RCP}} = \mathrm{S} \cup \mathrm{A}$ $\mathrm{S} = \{(\text{‘Active‘}, \sigma, \mathrm{t_{Last}}, \mathrm{t_{WCT}}) \mid \sigma \in \{0, 0.05\}, \mathrm{t_{Last}}, \mathrm{t_{WCT}} \in \mathbb{R}^{+}_{0,\infty}\}$ $\mathrm{s_{initial}} = (\text{‘Active‘}, 0, 0, 0)$ $\mathrm{A} = \{\mathrm{getWCT}() \to \mathbb{R}^{+}_{0,\infty}\}$
$\mathbf{Y}$	$\mathrm{Y} = \varnothing$
$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}}((\text{‘Active‘}, \sigma, \mathrm{t_{Last}}, \mathrm{t_{WCT}}))$: if $\mathrm{t_{Last}} - \mathrm{t_{WCT}} < 0.05$ then: $(\text{‘Active‘}, 0, \mathrm{t_{Last}}, \mathrm{getWCT}())$ else: $(\text{‘Active‘}, 0.05, \mathrm{getWCT}(), \mathrm{getWCT}())$
$\delta_{\mathbf{ext}}$	not used
$\delta_{\mathbf{con}}$	not used
λ	not used
$\mathbf{ta}$	$\mathrm{ta}((\text{‘Active‘}, \sigma, \mathrm{t_{Last}}, \mathrm{t_{WCT}})) := \sigma$

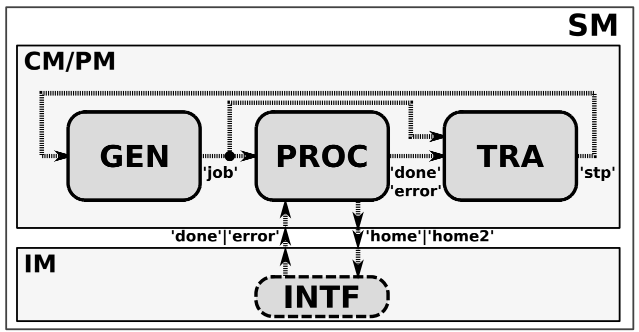
Fall 1: Wenn $(\mathrm{t_{WCT}} - \mathrm{t_{Last}}) < 0.05$ gilt, dann wird mit der Aktivität getWCT() der Wert von $\mathrm{t_{WCT}}$ aktualisiert und mit $\sigma = 0$ sofort ein neues internes Ereignis eingeplant.

Fall 2: Ansonsten wird mit getWCT() der Wert von $\mathrm{t_{WCT}}$ und $\mathrm{t_{Last}}$ aktualisiert sowie mit $\sigma = 0.05$ ein neues internes Ereignis als nächster Synchronisationspunkt eingeplant.

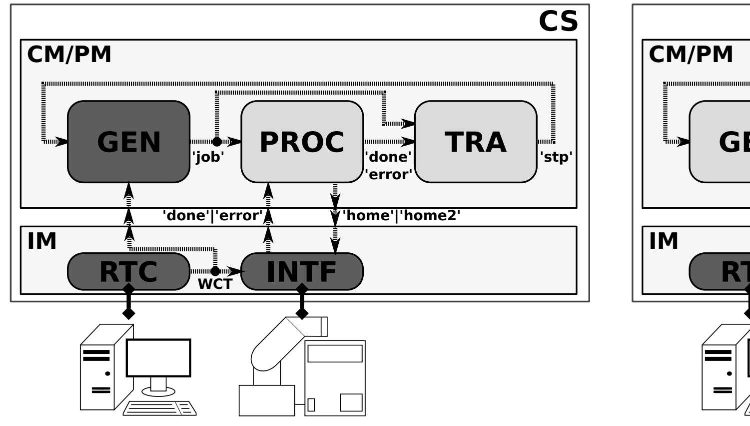
Unter der Voraussetzung, dass die unterliegenden Software- und Hardwareschichten die anwendungsbezogenen Echtzeitanforderungen erfüllen, kann mit PDEVS-RCP-V2 ein nahtloser Übergang vom SM zur SiL-Steuerung erfolgen. Im Anhang A wird eine Erweiterung zum Konzept der RTC disktuiert. Im nächsten Abschnitt wird die Nutzung von PDEVS-RCP-V2 im Kontext der durchgängigen Steuerungsentwicklung mit dem SBC-Ansatz an einem Beispiel dargestellt.

4.2 Beispiel zum SBC mit PDEVS-RCP-V2

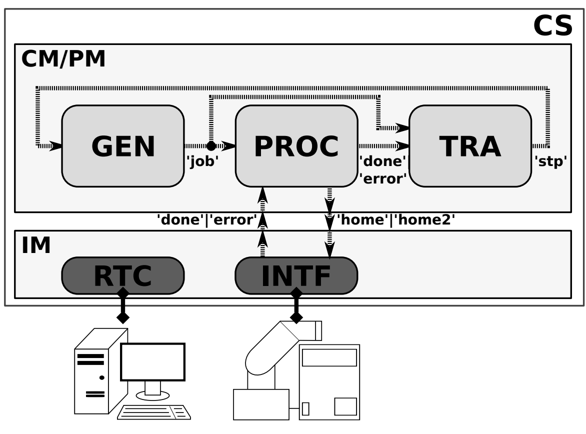
Das in Abbildung 4.4 gezeigte Beispiel ist angelehnt an Risco-Martín [90] und wird nachfolgend zur Diskussion von Prozessinteraktionen verwendet. Das Beispiel betrachtet ein ordinäres Single-Server-System und besteht in der Entwurfsphase aus einem GEN, einem PROC und einem TRA. Bezüglich der Terminies sei darauf verwiesen, dass diese keinen Bezug zu gleichnamigen Begriffen des später eingeführten Experimental Frames besitzen. Im operativen Betrieb soll der PROC mit einem realen Prozess interagieren. Teilbild (a) zeigt die Modellstruktur des Beispiels nach Übergang von der Entwurfs- zur Automatisierungsphase nach dem SBC-Ansatz. Auf Grund der geringen Problemkomplexität wurden das CM und das PM in einer Modellschicht abgebildet. Die Teilbilder (b) und (c) zeigen die Modellstrukturen nach Übergang des SM zu einer CS für den operativen Betrieb. Die Unterschiede bei der Umsetzung mit PDEVS-RCP nach Schwatinski [103] und PDEVS-RCP-V2 sind in Form der dunkelgrauen Blöcke dargestellt und werden nachfolgend disktutiert.



(a) SM in der Automatisierungsphase



(b) CS für operativen Betrieb mit PDEVS-RCP



(c) CS für operativen Betrieb mit PDEVS-RCP-V2

Abbildung 4.4: Anwendungsbeispiel nach SBC-Ansatz

Im spezifischen Anwendungsfall modelliert der PROC alternierende Bewegungsbefehle eines Roboters und muss im operativen Betrieb mit dem realen Roboter interagieren. Abbildung 4.5 zeigt die umzusetzende Bewegungsfolge: home $\rightarrow$ home2 $\rightarrow$ home $\rightarrow$ ....

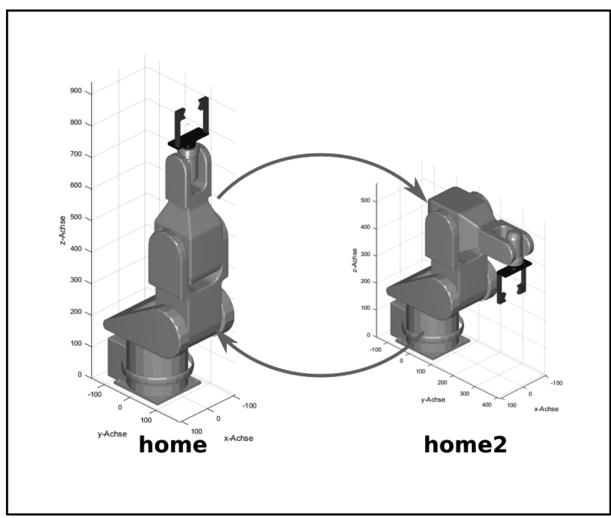


Abbildung 4.5: Vordefinierte Positionen home und home2

4.2.1 Übergang von der Entwurfs- zur Automatisierungsphase

In der Automatisierungsphase besteht das SM aus einem kombinierten CM/PM und einem IM. Das IM besteht aus einer atomaren Komponente INTF, welche die Schnittstelle zwischen PROC und realen Roboter simuliert. Nachfolgend wird das dynamische Verhalten der einzelnen Komponenten mit erweiterten DEVS-Diagrammen darsgestellt. Im Anhang B sind die DEVS-Diagramme in Kombination mit den PDEVS-Mengennotationen dargestellt.

Abbildung 4.6 zeigt das dynamische Verhalten des INTF. Das INTF empfängt vom PROC die Bewegungsbefehle als Eingangsereignisse und sendet Statusmeldungen als Ausgangsereignisse zurück. Die Ausführungszeit eines Bewegungsbefehls ist per Zufallswert modelliert ($\sigma = \text{random}(5, 10)$). Ist die Ausführungszeit größer einem Schwellwert ($\sigma > 9,5$) liegt ein Fehlerzustand vor. Je nach Ausführungsfall wird ein Ausgangsereignis 'done' oder 'error' generiert. Nach den Betrachtungen im vorangegangenen Abschnitt entspricht der Aufruf des Zufallszahlengenerators (Methode random) einer Prozessinteraktion mit der Laufzeitumgebung, analog der Methode getWCT bei der RTC.

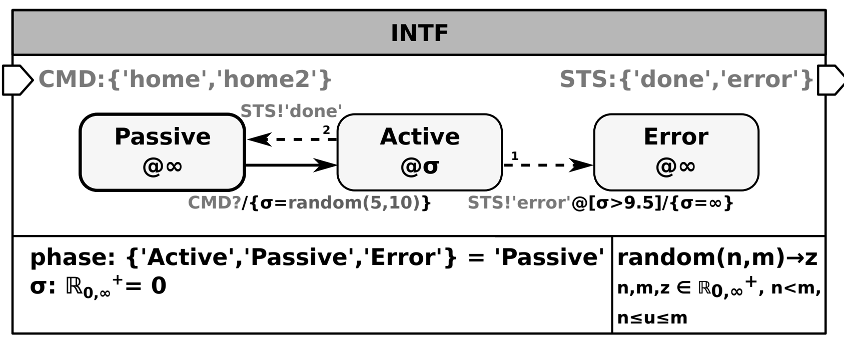


Abbildung 4.6: Erweitertes DEVS-Diagramm INTF in der Automatisierungsphase

Auf der CM/PM-Ebene generiert der GEN gemäß Abbildung 4.7 zufallsbasiert Aufträge ('job'), die als Eingangsereignisse an den PROC gesendet werden. Bei einem Eingangsereignis 'stp' wird die Auftragsgenerierung gestoppt und der GEN wechselt in den Zustand

Passive. Der PROC nach Abbildung 4.8 setzt Aufträge ('job') vom GEN in Bewegungsbefehle um und sendet diese als Ausgangsereignisse ('home','home2') an das INTF. Initial ist der PROC im Zustand Passive. Bei einem Eingangsereignis 'job' geht er in den Zustand Active über, verweilt dort bis eine Statusmeldung ('done','error') vom INTF erfolgt und wechselt in den Zustand Passive oder den Zustand Error. Eingehende Ereignisse 'job' werden im Zustand Active ignoriert. Der TRA, dargestellt in Abbildung 4.9, erhält vom GEN ebenfalls die Auftragsereignisse 'job' und vom PROC die Statusmeldungen 'done' beziehungsweise 'error'. Beim Eingangsereignis 'error' wechselt er in den Zustand Error und generiert ein Ausgangsereignis 'stp' für den GEN. Auf Basis der Eingangsereignisse 'job' und 'done' könnte der TRA gemäß der Theorie von Zeigler et al. [131] weitere *Values-of-Interest* berechnen, wie zum Beispiel die Anzahl abgelehnter Aufträge. In der gezeigten Spezifikation werden nur die 'job' sowie die 'done' Ereignisse aufsummiert.

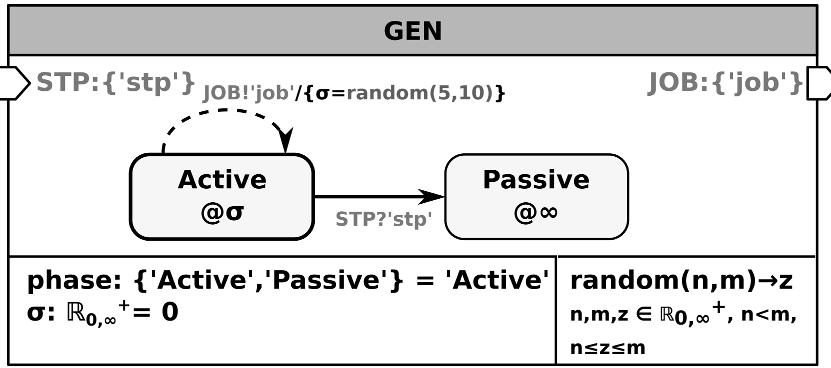


Abbildung 4.7: Erweitertes DEVS-Diagramm GEN

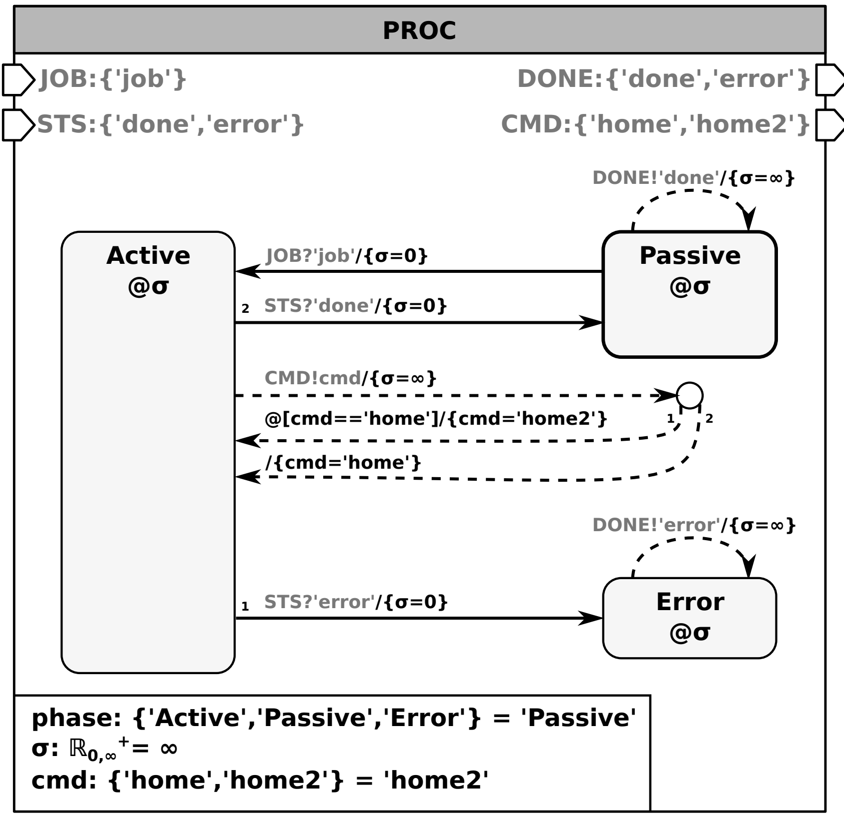


Abbildung 4.8: Erweitertes DEVS-Diagramm PROC

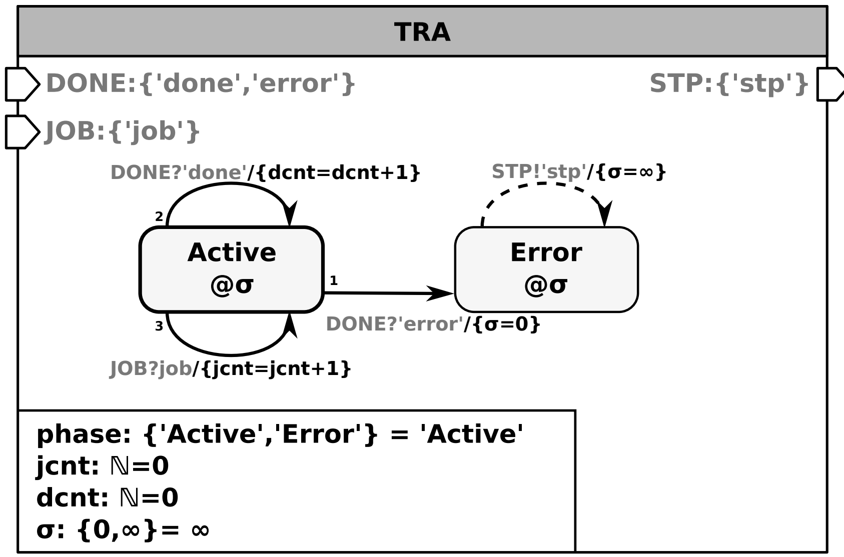


Abbildung 4.9: Erweitertes DEVS-Diagramm TRA

4.2.2 Übergang von der Automatisierungsphase zum operativen Betrieb mit PDEVS-RCP

Beim Übergang in den operativen Betrieb muss das SM so konkretisiert werden, dass es als CS mit einer Prozessumgebung interagieren kann. Die Umsetzung mit PDEVS-RCP nach Unterabschnitt 3.3.5 ist schematisch im Teilbild 4.4 (b) gezeigt. Alle dunkelgrauen Komponenten sind anzupassen und werden zu PDEVS-RCP Komponenten erweitert. Die neu hinzugekommene Komponente RTC modelliert eine Echtzeituhr, die die WCT zyklisch als Ausgangsereignis verschickt. Auffällig ist, dass nicht nur das INTF dieses Ereignis benötigt, sondern auch der GEN. Folglich muss GEN um einen neuen Eingangsport ergänzt werden. In der Entwurfsphase wurde die Dynamik des GEN auf Basis von VT entwickelt. PDEVS-RCP unterstützt keine unmittelbare Synchronisation zwischen VT und WCT, denn nach der Definition gilt: $\mathrm{ta}(s) \in \{0, \infty\}$. In der Betriebsphase muss GEN mit WCT arbeiten. Die Echtzeitsyncronisation des GEN erfolgt über die externen Eingangsereignisse der RTC. Die bereits getestete Komponente GEN muss modifiziert werden, obwohl sie keine Prozesskommunikation besitzt.

4.2.3 Übergang von der Automatisierungsphase zum operativen Betrieb mit PDEVS-RCP-V2

PDEVS-RCP-V2 behebt obiges Problem mit der neuen Spezifikation der Komponente RTC (Abschnitt 4.1.2, Abb. 4.3). Wie Teilbild 4.4 (c) zeigt, muss das SM auf IM-Ebene nur um die RTC gemäß Abbildung 4.3 ergänzt werden. Diese synchronisiert die VT und die WCT global. Es entfällt das zyklische Versenden der WCT in Form externer Ereignisse an zu synchronisierende Komponenten. Eine Modifikation des GEN ist nicht mehr erforderlich. Wie zuvor diskutiert, können gewöhnliche PDEVS und PDEVS-RCP-V2 Komponenten in einem Modell gemeinsam genutzt werden. Dementsprechend muss das INTF auf IM-Ebene gemäß Abbildung 4.10 konkretisiert werden. Der durch Zufallswerte simulierte

Bewegungsablauf, wird durch Aktivitäten zur Prozessinteraktion ersetzt und es werden zwei neue Zustandsvariablen (t_{CMD}, id) eingeführt. Die Aktivität getWCT arbeitet analog wie bei der RTC erklärt. Die Aktivität rmove startet eine neue Roboterbewegung und gibt eine id zur Identifikation der aktuellen Bewegung zurück. Auf Basis der id ermittelt die Aktivität ris, ob die Bewegung abgeschlossen ist. Mit Hilfe der Aktivität getWCT wird die Ausführungszeit einer Roboterbewegung überwacht. Beim Start der Bewegung, d.h. beim Zustandswechsel von Passive zu Active, wird die WCT auf die Zustandsvariable t_{CMD} geschrieben. Beim Beispiel wird angenommen, dass eine Bewegung mindestens 5 Sekunden ($\sigma = 5$) benötigt und nach maximal 10 Sekunden (getWCT $> t_{CMD} + 10$) abgeschlossen ist. Bei Überschreitung der oberen Zeitgrenze erfolgt ein Wechsel in den Zustand Error sowie das Versenden eines Ausgangsereignisses 'error'. Die Abtastzeit ist im Beispiel mit 0,1 Sekunden ($\sigma = 0.1$) festgelegt. Bei Einhaltung des vorgegebenen Zeitintervalls wird ein Ausgangsereignis 'done' versendet und in den Zustand Passive gewechselt.

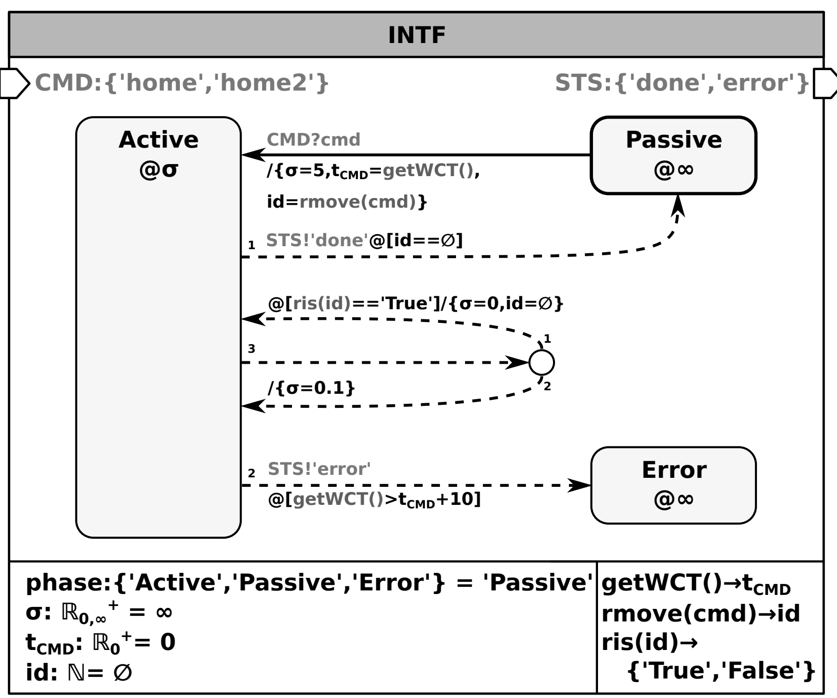


Abbildung 4.10: Erweitertes DEVS-Diagramm INTF im operativen Betrieb

4.2.4 Aufgabenorientierte Betrachtung

Ausgehend von den Betrachtungen zur Umsetzung aufgabenorientierter Steuerungen (Task-Oriented-Control, TOC) nach dem SBC-Ansatz, kann das betrachtete Beispiel, wie in Abbildung 4.11 gezeigt, strukturiert werden. Der GEN und der TRA repräsentieren die Aufgaben. Der PROC repräsentiert das Weltmodell und bildet im Zusammenspiel mit dem INTF den Aufgabentransformator. Die Dynamikspezifikation der atomaren Systeme entspricht jener im vorangegangen Abschnitt. Alternativ könnte auf der CM-Ebene noch eine formale Aufgabe *bearbeite* spezifiziert werden, welche ohne weitere Funktionalität nur als Verbindungsglied zum PROC agieren würde.

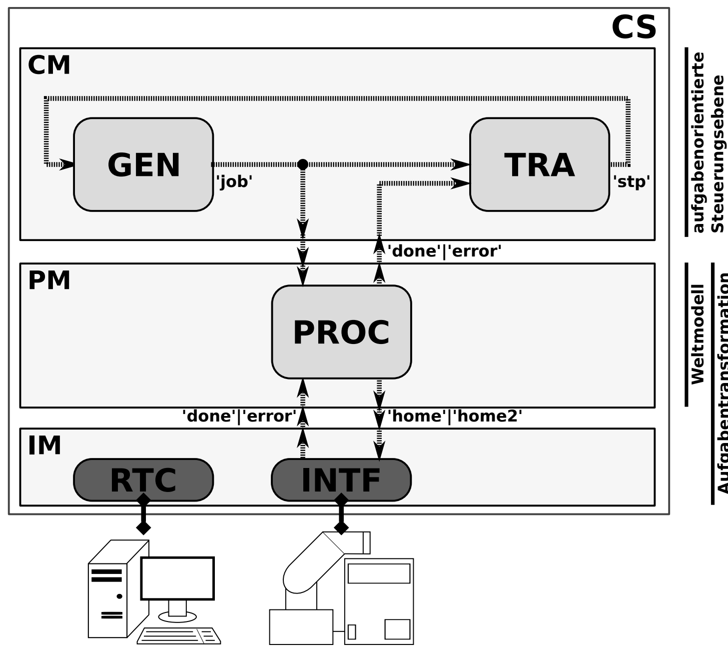


Abbildung 4.11: Beispiel im Kontext von SBC und TOC

In diesem Abschnitt wurde die SBC-basierte Steuerungsentwicklung mit PDEVS-RCP-V2 an einem Referenzbeispiel aufgezeigt. Weiterhin wurde der Bezug zu einer TOC-basierten Problemstellung für ein SRS hergestellt.

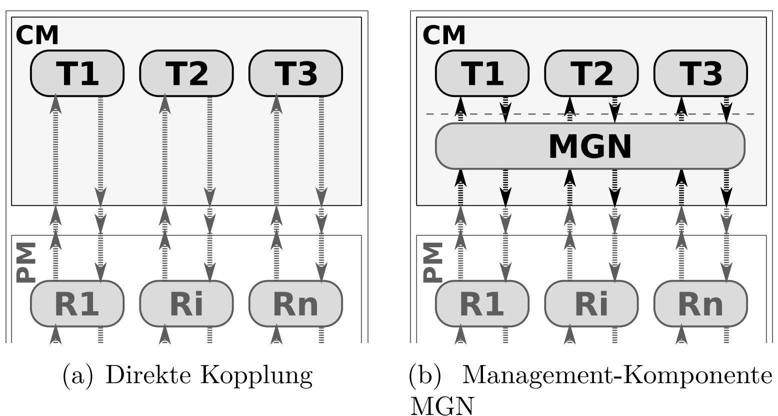
4.3 TOC basierte Lösungsansätze für MRS

In diesem Abschnitt werden Konzepte zur Umsetzung von TOC nach dem SBC-Ansatz für Multi-Robotersysteme (MRS) betrachtet. In Erweiterung zur TOC bei SRS müssen bei MRS zusätzlich Interaktionen zwischen den Robotern berücksichtigt werden. D.h. neben den Fragen *WAS* und *WIE*, ist zusätzlich die Frage des *WER* zu lösen. Es ist festzulegen, welcher Roboter welche Aufgaben löst. Weiterhin bedingen einige Interaktionen eine Kommunikation der Roboter zur Laufzeit. Ausgehend von den im Abschnitt 2.5.2 eingeführten Interaktionsklassen werden prinzipielle Lösungsansätze diskutiert. Dabei steht die Strukturierung der CM- und PM-Ebene im Vordergrund. Bezugnehmend auf Abschnitt 2.5.2 und Abbildung 2.20 wird das dort eingeführte Transportproblem als Referenzbeispiel genutzt. Weiterhin wird angenommen, dass die Roboter über ein integriertes Kamerasystem zur Identifikation und Positionsbestimmung von Bauteilen verfügen. Die konkrete Spezifikation ausgewählter Lösungsansätze mit PDEVS-RCP-V2 wird im Kapitel 6 gezeigt.

4.3.1 Interaktionsklasse 1 und 2

Eine grundlegende Eigenschaft der Interaktionsklasse 1 und 2 (Abb. 2.20) ist die Trennung der Arbeitsräume der Roboter. Hieraus folgt, dass jeder Roboter seine Aufgaben unabhängig vom anderen Robotern lösen kann. Damit ist keine Kommunikation der Roboter zur Laufzeit notwendig und die Aufgabenzuordnung (*WER*) ist oft durch die Arbeitsräume gegeben. Unterscheiden sich aber zum Beispiel die Aufgaben bei gleichen

Bauteilen, ist eine explizite Aufgabenzuordnung erforderlich. In Abbildung 4.12 sind zwei prinzipielle Varianten zur Aufgabenzuordnung auf der CM-Ebene dargestellt. Teilbild (a) zeigt die Zuordnung von Aufgaben Ti zu Robotern Ri durch direkte Kopplungen. Bei der Variante in Teilbild (b) ist die CM-Ebene um eine Komponente Management MGN erweitert. Diese verfügt über Wissen zur Anzahl, Leistungsfähigkeit, Position und aktuellen Zuständen, der mit ihr gekoppelten Roboter der PM-Ebene. Die Hauptaufgabe von MGN ist die Zuordnung von Aufgaben zu Robotern und damit das Lösen der Frage *WER*. Ein Problem dieser Lösungsvariante ist die hohe Komplexität und damit die schlechte Skalierbarkeit. Weiterhin sollte die CM-Ebene entsprechend dem TOC-Ansatz ausschließlich durch Aufgaben spezifiziert sein. Aus diesem Grund wird für Interaktionsklasse 1 und 2 ein Lösungsansatz auf Basis der direkten Zuordnung von Aufgaben zu Robotern analog Teilbild (a) vorgeschlagen.



(a) Direkte Kopplung

(b) Management-Komponente MGN

Abbildung 4.12: Varianten zur Zuordnung von Aufgaben (Ti) zu Robotern (Ri) durch die CM-Ebene

Wie in Abbildung 4.13 dargestellt, legt die Steuerung explizit fest welche Aufgaben durch welchen Roboter abzuarbeiten sind. Die Notation Ri.CMD steht für Steuerkommandos, die von der CM-Ebene an einen Roboter Ri gesendet werden, und Ri.STS steht für Statusmeldungen, die von einem Roboter Ri an die CM-Ebene gesendet werden. Auf der CM-Ebene können die Aufgaben roboterspezifisch als nebenläufig auszuführende Aufgabensequenzen beschrieben werden. Jeder Roboter hat zunächst die Aufgabe, die Position Pos eines ihm zugeordneten Bauteils im Eingangspuffer (Input-Buffer, IB) zu bestimmen (Pos=IdPrt(IB,A oder B)). Der Roboter R1 händelt Teile vom Typ A und der Roboter R2 Teile vom Typ B. Nach der Identifikation bewegt sich der jeweilige Roboter in eine geeignete Ausgangsposition Move(Pos), um das identifizierte Bauteil zu greifen (PickPrt). Danach wird das Bauteil zum Ausgangspuffer (Output-Buffer, OB) transportiert (Move(OB)) und dort abgelegt (PlacePrt). Solange weitere Bauteile verfügbar sind, wird die Aufgabensequenz wiederholt.

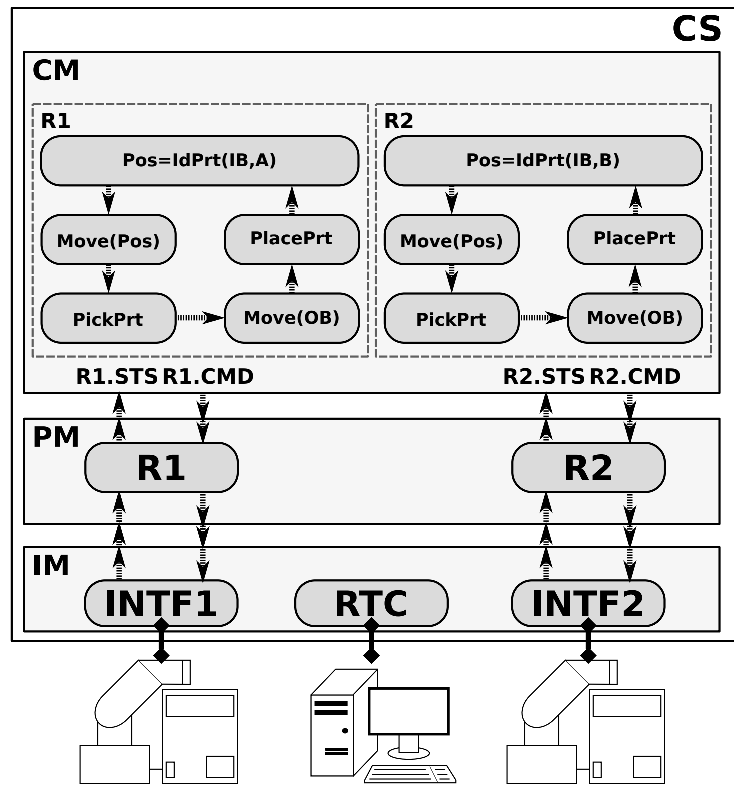


Abbildung 4.13: Lösungsansatz für Interaktionsklasse 1 und 2

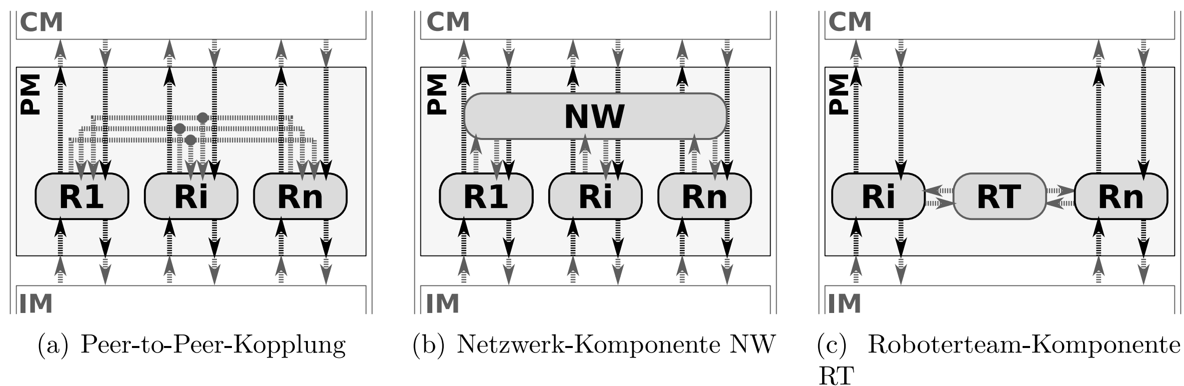
Aus Gründen der Übersichtlichkeit wird der Aspekt der Nebenläufigkeit im Folgendem nur auf der CM-Ebene explizit dargestellt. Die nebenläufige Ausführung von Modellkomponenten ist gemäß Abschnitt 3.2.2 durch die PDEVS-Ausführungsalgorithmen gewährleistet.

4.3.2 Interaktionsklasse 3

Die Interaktionsklasse 3 beinhaltet eine zeitliche Koordination von Robotern zur Abstimmung bestimmter Aufgaben. Die Anforderung folgt aus gemeinsamen Arbeitsräumen. Um Kollisionen zu vermeiden, müssen Bewegungen zeitlich abgestimmt werden.

Lösungsvarianten auf PM-Ebene: Abbildung 4.14 zeigt drei Varianten zur bidirektionalen Kommunikation der Roboter auf der PM-Ebene. In Teilbild (a) wird die Kommunikation auf Basis von direkten Kopplungen zwischen den Roboterkomponenten gelöst. Die durch die Kopplungen entstehende Struktur ähnelt einem Peer-to-Peer-Netzwerk, wie es aus dem Bereich der Netzwerktechnik bekannt ist. Der Aufwand Komponenten in dieser Weise miteinander zu verkoppeln, steigt stark mit ihrer Anzahl. Aus diesem Grund wird mit Teilbild (b) die Einführung einer neuen Komponente Netzwerk NW vorgeschlagen. Diese dient als *Mittelsmann* zwischen den Roboterkomponenten. Wie aus der Abbildung ersichtlich ist, reduziert sich die Anzahl der zur Kommunikation notwendigen Ein- und Ausgangsports je Roboterkomponente R auf einen Ein- und Ausgangsport. Die Struktur der Komponente Ri ist unabhängig von der Anzahl miteinander kommunizierender Komponenten. NW soll keine roboterspezifischen Informationen speichern. Folglich müssen die Roboterkomponenten im Fall einer Interaktion mit anderen Komponenten selbst alle relevanten Informationen speichern. Dies führt zu einem hohen und dezentralen Informati-

onsvolumen. An diesem Punkt setzt die Komponente Roboter-Team RT aus Teilbild (c) an. Im Gegensatz zur Komponente NW speichert RT alle für das Team relevanten Informationen. Durch die zentrale Komponente zur Abstimmung der Roboter reduzieren sich die Komplexität der Roboterkomponenten Ri und der Kommunikationsaufwand.



(a) Peer-to-Peer-Kopplung (b) Netzwerk-Komponente NW (c) Roboterteam-Komponente RT

Abbildung 4.14: Lösungsvarianten auf PM-Ebene

Beispiel zur Interationsklasse 3: Beim Beispiel nach Abbildung 2.20 muss sichergestellt werden, dass die Roboter nicht zeitgleich auf den Arbeitsraum des IB zugreifen. Abbildung 4.15 zeigt einen Lösungsansatz zur Umsetzung einer Steuerung für dieses Beispiel.

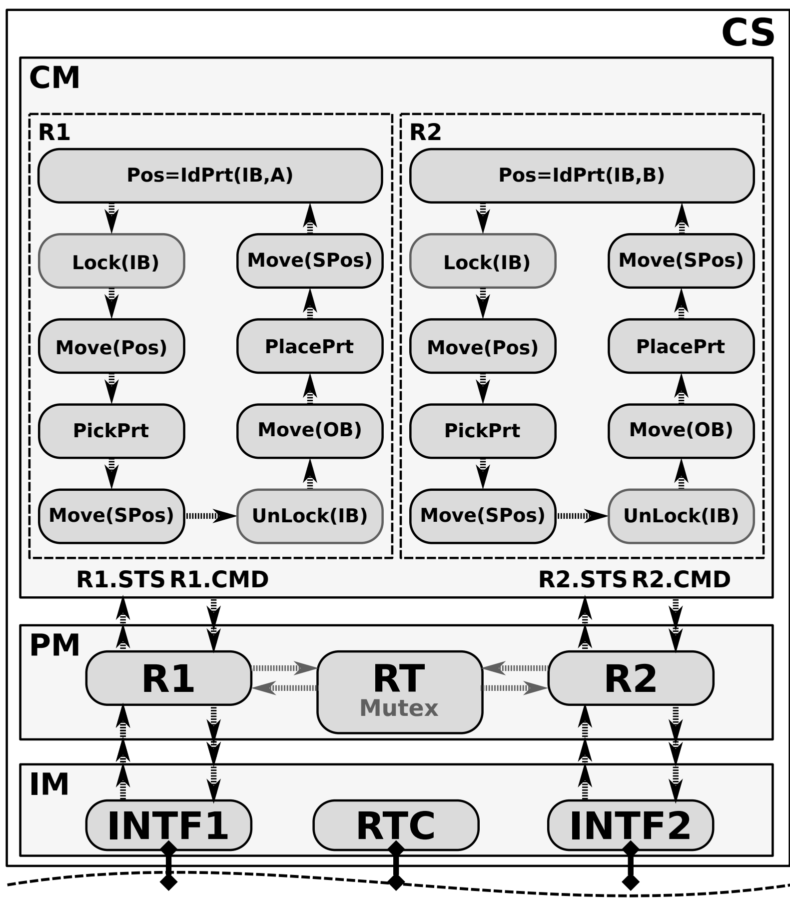
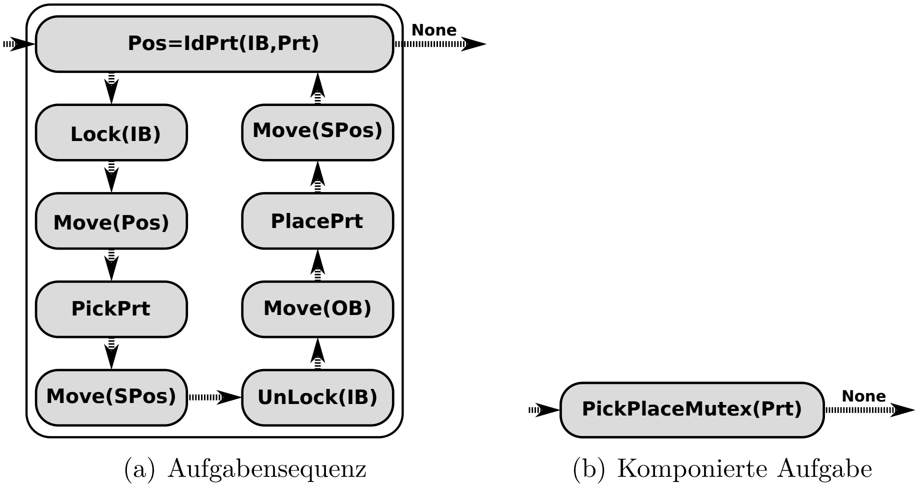


Abbildung 4.15: Lösungsansatz für Interaktionsklasse 3

Um einen wechselseitigen Ausschluss (Mutex) bei der Ausführung der Aufgabe Move(IB) für beide Roboter abzubilden, werden die zwei parametrierbaren Aufgaben Lock und UnLock eingeführt. Diese spezifizieren den Anfang und das Ende einer Aufgabenkoordination auf der CM-Ebene (das *WAS*). Zusätzlich wird die PM-Ebene um die zuvor diskutierte Komponente RT erweitert, welche der Kommunikation zwischen den Roboterkomponenten R1 und R2, zur Abbildung gemeinsamer Zustände und zur Umsetzung der Aufgabenkoordination durch Mutex (dem *WIE*) dient.

Umsetzung des Mutex: Das vorgestellte Konzept wurde in Anlehnung an die Interprozesskommunikation nach Tannenbaum und Bos [114] entwickelt. Beide Roboter müssen die gleiche Aufgabensequenz mit den in rot hervorgehobenen Aufgaben Lock und UnLock ausführen. Beim Aufruf der Aufgabe Lock wird eine Ressource mit dem Namen IB vom jeweiligen Roboter reserviert und für andere blockiert. Über die Kopplungsrelationen wird die Information, welcher Roboter welche Ressource besitzt, in der RT-Komponente hinterlegt. Da eine Ressource beim Prinzip Mutex nur einen Besitzer haben kann, erhält der Roboter, der die Aufgabe Lock zuerst ausführt, ihren Besitz. Der andere Roboter kann die Aufgabenausführung Lock(IB) zunächst nicht abschließen. Jedoch kann der Wunsch die Ressource als nächster zu besitzen, bereits in RT hinterlegt werden. Die Reservierung wird erst gültig, wenn die Ressource mit der Aufgabe UnLock wieder freigegeben ist. Liegt dann bereits eine Reservierung vor, so geht der Besitz unmittelbar an den reservierenden Roboter über. Bevor ein Roboter, welcher im Besitz einer Ressource ist, diese wieder freigibt, sollte der kritische Arbeitsbereich verlassen sein. Hierfür wird die Aufgabe Move(SPos) definiert. Diese entspricht der Bewegung auf eine sichere Position.

Aufgabenkomposition: Zur Reduzierung der Komplexität und der Aufgabenwiederverwendung wird die in Teilbild 4.16 (a) gezeigte Aufgabensequenz durch Komposition zu einer neuen Aufgabe PickPlaceMutex zusammengefasst. Diese ist in Teilbild (b) dargestellt und verfügt über die selben Schnittstellen. Wie anhand der Namensgebung zu erkennen ist, definiert die Aufgabe den Transport von Bauteilen vom IB zum OB einschließlich gegenseitigem Ausschluss. Wenn ein gewünschtes Bauteil nicht identifiziert werden kann, wird die Aufgabensequenz abgebrochen und das Ereignis None ausgegeben.



(a) Aufgabensequenz (b) Komponierte Aufgabe

Abbildung 4.16: Die Aufgabe PickPlaceMutex

4.3.3 Interaktionsklasse 4

Bei der Interaktionsklasse 4 wird in Abbildung 2.20 eine neue Teileart C eingeführt. Der Transport der neuen Teileart erfordert die direkte Zusammenarbeit beider Roboter. Diese müssen ihre Bewegungsabläufe synchronisieren, um Bauteile des Typs C gemeinsam vom IB zum OB zu transportieren. Wie in Freymann et al. [39] gezeigt ist, kann der gemeinsame Transport beispielsweise mittels *geometrischer Kopplung* der Roboter erfolgen. Hierbei erfolgt die Positionsermittlung und Bewegungssteuerung eines Roboters direkt aus der Position und Bewegung eines anderen Roboters. Ein Lösungsansatz zur Umsetzung von Interaktionsklasse 4 ist in Abbildung 4.17 dargestellt. Wesentliche Änderungen gegenüber dem Lösungsansatz der Interaktionsklasse 3 in Abbildung 4.15 sind farblich hervorgehoben.

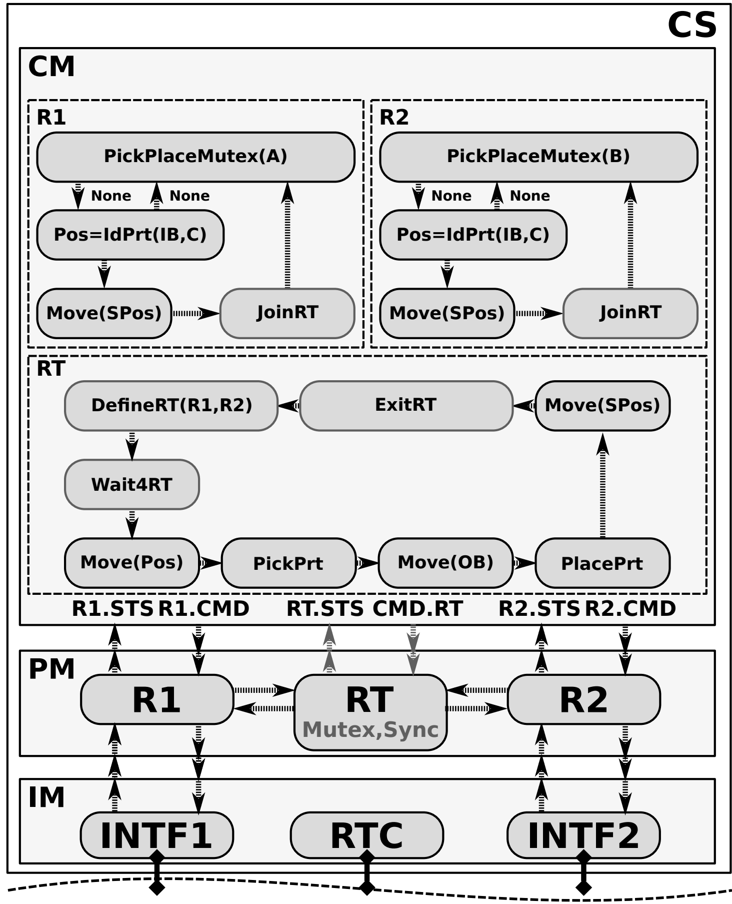


Abbildung 4.17: Lösungsansatz für Interaktionsklasse 4

Umsetzung der Synchronisation: Beide Roboter verfügen über identische Aufgabensequenzen, wobei sich lediglich die Parametrierung unterscheidet. Zunächst sollen durch die Roboter alle Bauteile vom IB zum OB transportiert werden, die nicht vom Typ C sind (Aufgabe PickPlaceMutex). Hierbei ist jeder Roboter auf eine Teileart spezialisiert. Der Zugriff auf den IB erfolgt durch gegenseitigen Ausschluss (Mutex). Sind die individuell lösbaren Aufgaben abgeschlossen, erfolgt eine Überprüfung des IB auf Teile vom Typ C und eine Positionsbestimmung (Pos=IdPrt(IB,C)). Wird durch einen Roboter ein entsprechendes Bauteil identifiziert, verfährt der Roboter auf eine geeignete Ausgangsposition (Move(SPos)) und tritt dem Team bei (JoinRT). Dies signalisiert die Bereitschaft zur Interaktion. Wenn beide Roboter ihre Bereitschaft signalisiert haben, wird mit DefineRT

und Wait4RT eine neue Aufgabensequenz, welche dem Team RT zugeordnet ist, ausgeführt. Die Komponente RT auf PM-Ebene synchronisiert (Sync) den gemeinsamen Transport eines Bauteils C (Aufgabentransformation). Der Zugriff auf die INTF-Komponenten der Roboter erfolgt indirekt über R1 und R2 auf PM-Ebene. Wenn das gemeinsame Platzieren durch RT abgeschlossen ist, wird die individuelle Aufgabenabarbeitung der Roboter erneut angestoßen (ExitRT(R1,R2)).

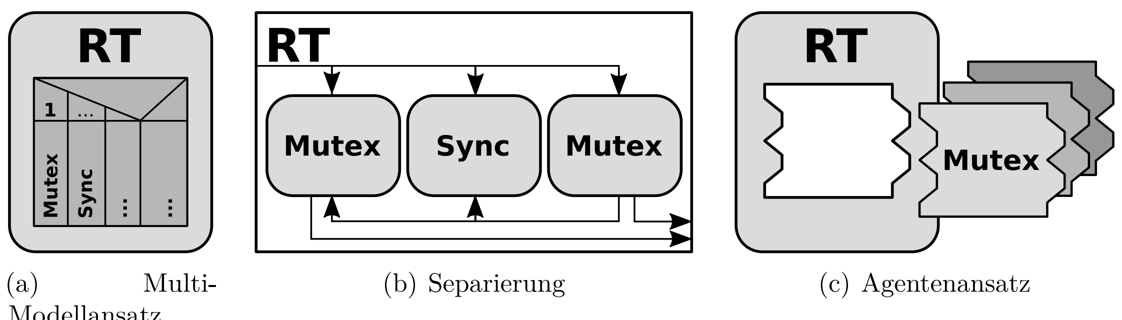
Varianten zur Umsetzung des RT auf PM-Ebene: Für Interaktionsklasse 4 muss die Komponente RT den gegenseitigen Ausschluss gemeinsamer Ressourcen (Mutex) und die Bewegungssynchronisation (Sync) mehrerer Roboter umsetzen. Damit steigt die Komplexität der Komponente. Nachfolgend werden drei Varianten zur Umsetzung von RT diskutiert. Abbildung 4.18 zeigt die prinzipiellen Ansätze der einzelnen Varianten.

Die Variante (a) basiert auf dem Multi-Modellansatz, wie zum Beispiel in Fishwick [35] diskutiert. Die Transformation der Interaktionsprinzipien (Mutex, Sync, ...) wird gemäß der Strukturierung eines Multi-Modells in einer atomaren Komponente RT abgebildet. Trotz der Strukturierung als Multi-Modell entsteht bei diesem Ansatz schnell eine komplexe atomare Komponente, die schwer testbar und wartbar ist. Weiterhin sind die umsetzbaren Interaktionsprinzipien fest kodiert und nicht zur Laufzeit der Steuerung austauschbar. Aufgrund der Umsetzung als atomare Komponente ist die Kommunikation zwischen einzelnen Interaktionsprinzipien relativ einfach realisierbar.

Bei Variante (b) wird jedes Interaktionsprinzip (Mutex, Sync) als separate atomare Komponente modelliert. Die Komponente RT besteht je nach Anwendungsfall aus nur einer atomaren Komponente oder aus einer Komposition atomarer Komponenten. Die Separierung der Interaktionsprinzipien in unterschiedliche atomare Komponenten reduziert die Komplexität der einzelnen Komponenten, erleitert den Funktionstest und die Wartbarkeit. Allerdings müssen kompatible Schnittstellen definiert werden, um eine Komposition von Interaktionen zu gewährleisten. Die umsetzbaren Interaktionsprinzipien sind über die Komponenten und deren Kopplungsrelationen fest kodiert und können zur Laufzeit nicht verändert werden. Nur bei Verwendung eines strukturdynamischen Simulators gemäß Zeigler [131] oder Hagendorf [45] könnten Interaktionsprinzipien zur Laufzeit ausgetauscht oder ergänzt werden. Die derzeitigen strukturdynamischen DEVS-Simulatoren unterstützen allerdings keine Echtzeit- und Prozessanbindung. Da der Rechenaufwand bei strukturdynamischen Simulatoren ansteigt, besitzen sie ein schlechteres Laufzeitverhalten.

Die dritte Variante ist in Teilbild (c) dargestellt, welche sich an Agentenansätzen, wie zum Beispiel in Yilmaz [128], orientiert. Mit dem Ansatz wird versucht, die Vorteile der beiden vorangegangenen Ansätze zu kombinieren. Die Zielstellung besteht in der Implementierung einer generischen atomaren RT Komponente, die unterschiedliche Interaktionsprinzipien umsetzen kann und zur Laufzeit dynamisch erweiterbar ist. Die generische Komponente definiert kein Interaktionsprinzip, sondern nur ein externes und internes Interface. Das externe Interface definiert die Ein- und Ausgangsports der RT Komponente für die PM-Ebene. Das interne Interface definiert die Schnittstelle zur Integration von Interaktionsprinzipien in Form dynamischer Interaktionsobjekte. Die einzelnen Interaktionsprinzipien oder Kombinationen von Interaktionsprinzipien sind als separate Komponenten, Interaktionsobjekte genannt, implementiert und in einer Modellbibliothek (MB) organisiert. Sie spezifizieren das dynamische Verhalten von Interaktionsprinzipien mit den PDEVS typischen Dynamikfunktionen $\delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, \text{ta}$. Zur Laufzeit empfängt die generische Komponente RT vom CM ein Ereignis zur Umsetzung eines speziellen Interaktionsprinzips. Durch die δ_{ext}

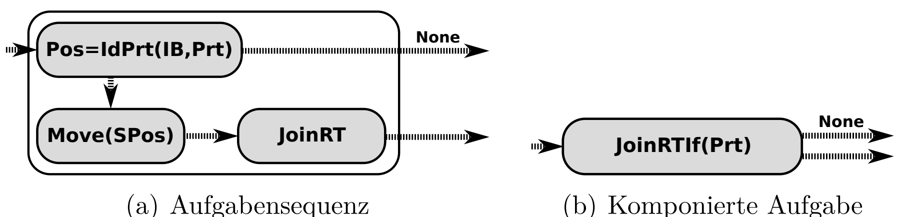
beziehungsweise δ_{con} Funktion der generischen Komponente RT wird das anzuwendende Interaktionsprinzip als Objekt aus der MB geladen und im Zustand S der Komponente RT gespeichert. Die Dynamikfunktionen der generischen Komponente RT greifen über das im Zustand $s \in S$ gespeicherte Interaktionsobjekt auf dessen Dynamikfunktionen zu. Mit dieser Lösung kann das CM das dynamische Laden notwendiger Methoden zur Laufzeit unter Nutzung eines nicht-strukturdynamischen DEVS-Simulators bewirken. Die MB kann um neue Interaktionsmethoden erweitert werden. An dieser Stelle sei aber vermerkt, dass die vordefinierte externe Schnittstelle der generischen Komponente RT und die Kopplungsrelationen auf der PM-Ebene fix sind, woraus Einschränkungen resultieren. Die Variante (c) wurde durch den Autor in [38] konkretisiert.



(a) Multi-Modellansatz (b) Separierung (c) Agentenansatz

Abbildung 4.18: Varianten zur Umsetzung des RT auf PM-Ebene

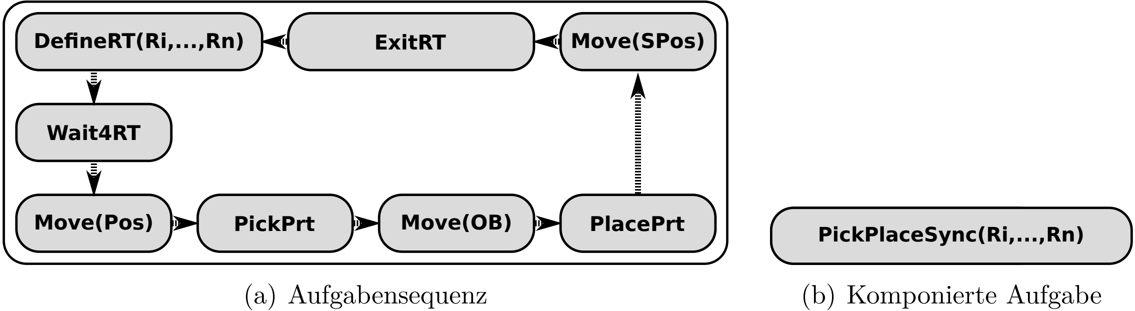
Aufgabenkomposition: Analog dem Vorgehen bei der Interaktionsklasse 3 ist es für die Wiederverwendung der Aufgaben sinnvoll, diese zu komponieren, wie in Abbilung 4.19 und Abbildung 4.20 gezeigt. Die Aufgabe JoinRTIf(Prt,Ri) definiert unter welcher Bedingung ein Roboter dem Team beitritt. Durch den Teambeitritt erfolgt die Steuerung des Roboters nicht mehr individuell, sondern koordiniert durch die RT-Komponenten auf CM- und PM-Ebene.



(a) Aufgabensequenz (b) Komponierte Aufgabe

Abbildung 4.19: Die komponierte Aufgabe JoinRTIf

Die Aufgabe PickPlaceSync dient zur Synchronisation (Sync) der Roboter beim gemeinsamen Transport eines Bauteils. Weiterhin definiert sie, wann die Roboter ihre individuelle Kontrolle (WakeUp(R1,R2)) wiedererlangen.



(a) Aufgabensequenz (b) Komponierte Aufgabe

Abbildung 4.20: Die komponierte Aufgabe PickPlaceSync

4.3.4 Interaktionsklasse 5

In Erweiterung zur Interaktionsklasse 4 wird in Interaktionsklasse 5 (Abb. 2.20) die Überlastung beziehungsweise Nichtauslastung eines Roboters durch eine ungleiche Verteilung der Teilearten thematisiert. Ein Roboter soll einen anderen Roboter unterstützen, falls dieser seine Aufgaben erledigt hat. Folglich müssen die Roboter über hinreichend geeignete Werkzeuge verfügen, um die unterschiedlichen Bauteile handeln zu können. Ein möglicher Lösungsansatz wird in Abbildung 4.21 gezeigt.

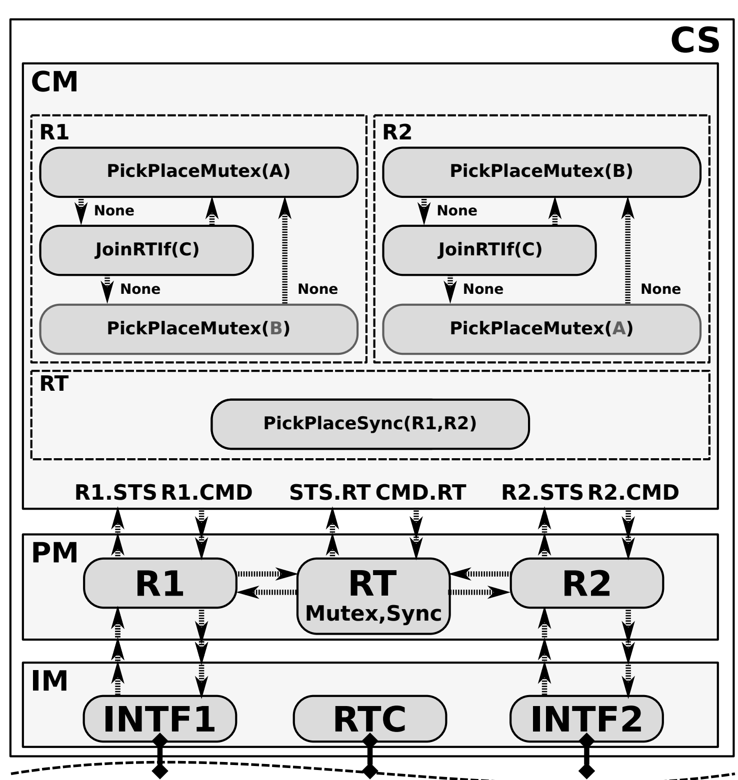


Abbildung 4.21: Lösungsansatz für Interaktionsklasse 5

Der Ansatz verwendet die parametrierbare Aufgabe PickPlaceMutex aus Abschnitt 4.3.2. Zunächst spezifiziert die Aufgabe PickPlaceMutex, dass alle Bauteile, die die Roboter eigenständig handeln können, vom IB zum OB transportiert werden. Anschließend beschreibt die Aufgabe JoinRTIf, dass die gemeinsam zu transportierenden Teile vom Typ

C zu transportieren sind. Sind diese beiden Aufgaben erledigt, beschreibt die diametral parametrierte Aufgabe PickPlaceMutex die Unterstützung des überlasteten Roboters, indem der nicht ausgelastete Roboter die "fremde"Teileart handelt. Der Zugriff auf den IB muss dazu abermals koordiniert erfolgen.

4.3.5 Interaktionsklasse 6

Bei der Interaktionsklasse 6 wird eine weitere Teileart D eingeführt (Abb. 2.20), welche nicht durch das aktuell konfigurierte Roboterteam transportiert werden kann. Somit muss das Team zum Handling der Teileart D temporär modifiziert werden. Dies kann entweder durch die Rekonfiguration eines Teammitglieds oder durch die temporäre Integration eines weiteren Roboters ins Team erfolgen.

Variante 1: Ein Lösungsansatz zur Rekonfiguration eines Roboters ist in Abbildung 4.22 gezeigt. Die parametrierbare Aufgabe Reconfig bildet den Tausch des Greiferwerkzeuges ab.

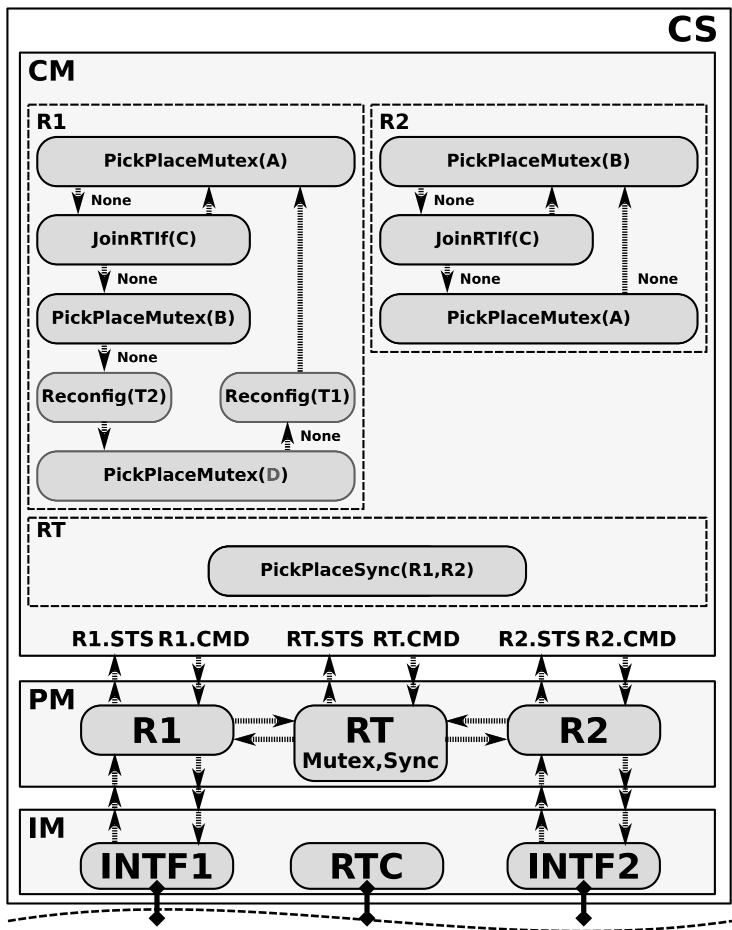
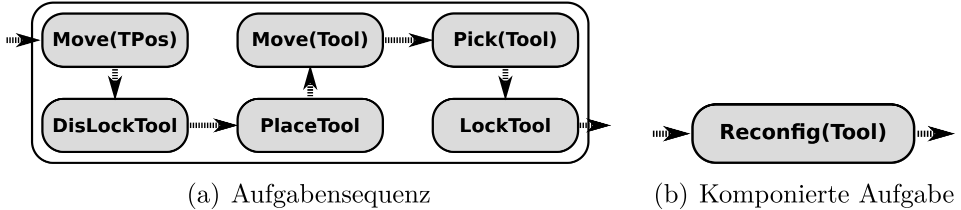


Abbildung 4.22: Erster Lösungsansatz für Interaktionsklasse 6 (Werkzeugtausch)

Die Struktur des Lösungsansatzes entspricht im Wesentlichen dem Lösungsvorschlag von Interaktionsklasse 5. Wenn der Roboter R1 alle Bauteile vom Typ A, die gemainsam zu handelden Teile C und gegebenenfalls unterstützend Teile vom Typ B transportiert hat, beschreibt die Aufgabe Reconfig(T2) den Wechsel des Greiferwerkzeuges. Mit dem

neuen Werkzeug sind gemäß Aufgabe PickPlaceMutex(D) die Teile vom Typ D zu transportieren. Nach Abschluss dieser Aufgabe beschreibt Reconfig(T1) den Rückwechsel des Greiferwerkzeuges.

Die Aufgabe Reconfig kann, wie in Abbildung 4.23 dargestellt, als eine Aufgabensequenz modelliert werden. Zunächst verfährt der Roboter auf eine zur Werkzeugablage geeignete Position (Move(TPos)). Anschließend wird das aktuelle Werkzeug gelöst (DisLookTool) und plaziert (PlaceTool). Nachfolgend bewegt sich der Roboter auf eine zur Aufnahme des neuen Werkzeuges geeignete Position (Move(Tool)) und greift dieses (Pick(Tool)). Das neue Werkzeug wird verriegelt (LockTool) und die Rekonfiguration ist abgeschlossen.



(a) Aufgabensequenz (b) Komponierte Aufgabe

Abbildung 4.23: Komposition der Aufgabe Reconfig

Variante 2: Abbildung 4.24 zeigt einen Lösungsansatz für die temporäre Unterstützung des Roboterteams durch einen Roboter R3. Dieser tauscht mit R1 seine Aktivität (Aufgabe SwapRole). Das heißt, jeweils nur einer der beiden Roboter ist zu einem Zeitpunkt aktiv.

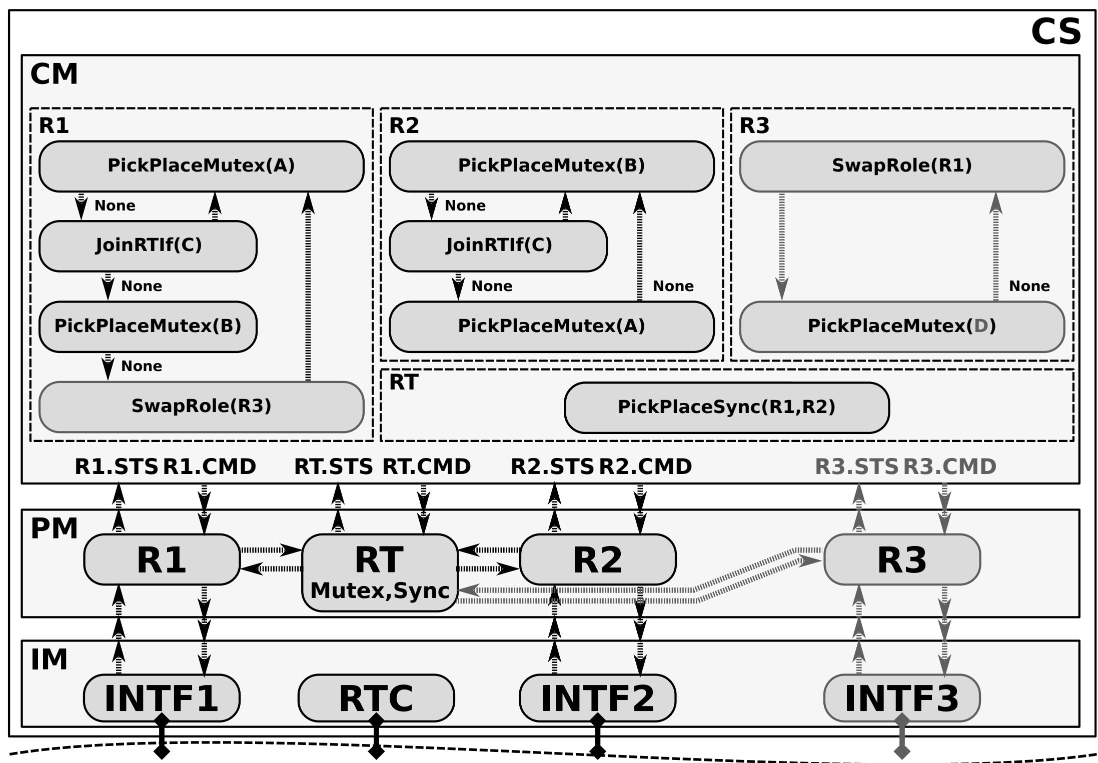
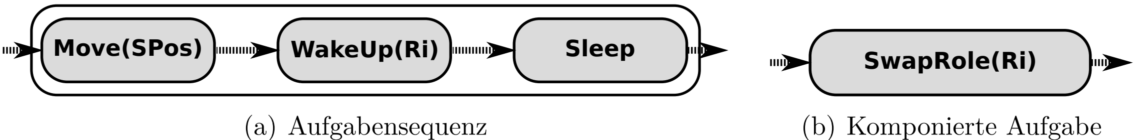


Abbildung 4.24: Zweiter Lösungsansatz für Interaktionsklasse 6 (temporärer Roboter)

Wie in Abbildung 4.25 dargestellt ist, kann die parametrierbare Aufgabe SwapRole(Ri) als eine Aufgabensequenz realisiert werden. Diese definiert zuerst eine Bewegung des

aktuell aktiven Roboters auf eine sichere Position (Move(SPos)). Anschließend wird mit der Aufgabe WakeUp(Ri) der Roboter Ri aktiviert und der bisher aktive Roboter mit der Aufgabe Sleep in einen Wartezustand versetzt.



(a) Aufgabensequenz (b) Komponierte Aufgabe

Abbildung 4.25: Komposition der Aufgabe SwapRole

Die aus dem Rollentausch der Roboter resultierenden Änderungen auf der PM- und IM-Ebene sind in Abildung 4.24 farblich hervorgehoben dargestellt. Bereits der konzeptionelle Lösungsansatz lässt erahnen, dass die Komplexität der Aufgabentransformation zunimmt. Drei Varianten zur Umsetzung komplexer Aufgabentransformationen wurden bereits beim Lösungsansatz der Interaktionsklasse 4 in Abschnitt 4.3.3 diskutiert. Prinzipiell sind zeitdynamische Strukturänderungen mit den in Abschnitt 4.3.3 diskutierten Methoden abbildbar. Die Abbildung derartiger Strukturdynamik auf Systemmodellebene erhöht die Modellkomplexität oft beträchtlich und erschwert damit die Modellwartung. Aus diesem Grund wird im Kapitel 5 ein alternativer Modellierungsansatz eingeführt, bevor in Kapitel 6 detaillierte Umsetzungen mit PDEVS-RCP-V2 diskutiert werden.

4.4 Zusammenfassung

Im Kapitel wurde die Entwicklung aufgabenorientierter Steuerungen auf Basis des SBC-Ansatzes und des DEVS-Formalismus untersucht. Zuerst wurden, basierend auf der Analyse der DEVS-Formalismen in Abschnitt 3.3, die Probleme bestehender DEVS-Ansätze hinsichtlich einer durchgehenden Steuerungsentwicklung nach dem SBC-Ansatz herausgearbeit und ein modifizierter PDEVS-Ansatz entwickelt. Der neue Ansatz wurde PDEVS-RCP-V2 genannt und ist eine Weiterentwicklung des PDEVS-RCP-Ansatzes. PDEVS-RCP-V2 erleichtert die Nutzung und Weiterentwicklung von DEVS-Modellen beim Übergang von der Entwurfsphase in die Automatisierungsphase beziehungsweise in den operativen Steuerungsbetrieb. Anhand eines häufig verwendeten Fallbeispiels wurden die Vorteile von PDEVS-RCP-V2 zur Vorgängerversion aufgezeigt.

Wie bereits in Abschnitt 2.4 dargestellt, unterstützt der SBC-Ansatz prinzipiell eine aufgabenorientierte Steuerungsentwicklung. In diesem Kapitel wurde untersucht, ob auch Interaktionen zwischen Robotern in einem MRS aufgabenorientiert beschrieben werden können. Die Untersuchung erfolgte anhand der in Abschnitt 2.5 betrachteten Interaktionsklassen. Es wurde für jede Klasse konzeptionell eine aufgabenorientierte Spezifikation entwickelt. Dabei zeigte sich, dass bekannte Prinzipien zur Interprozesskommunikation bei Betriebssystemen sehr gut auf MRS übertragbar sind.

Der Lösungsansatz zur sechsten Interaktionsklasse zeigt, dass die Steuerungsspezifikation und die Aufgabentransformation aufgrund der Flexibilität der Steuerung eine Herausforderung für Entwickler darstellt. Das Problem der Komplexität flexibler Steuerungen wurde bereits in Maletzki [67] im Kontext von SRS diskutiert. Als Lösungsvorschlag wurde in

Maletzki das SES/MB Framework mit Bezug auf aufgabenorientierte Steuerungen und den SBC-Ansatz untersucht.

Im nächsten Kapitel werden kurz die Grundlagen des SES/MB Frameworks eingeführt und konzeptionell deren Anwendung im Kontext dieser Arbeit betrachtet.

5 Flexible aufgabenorientierte Multi-Robotersteuerungen

Die Charakteristik einer flexiblen Steuerung besteht in Anlehnung an Weller [125] in der situativen Anpassung der Steuerung an die aktuellen Prozessgegebenheiten. Das heißt, eine flexible Steuerung ist hinsichtlich ihrer Parameter oder Struktur konfigurierbar. Eine Steuerungskonfiguration wird nachfolgend auch als Steuerungsvariante bezeichnet. Eine Steuerungsvariante besitzt eine endliche Gültigkeit. Sie kann zu einem bestimmten Zeitpunkt zugunsten einer anderen Variante verworfen werden. Jede Variante besitzt eine begrenzte Funktionalität zur Lösung eines Teilproblems. In diesem Kontext löst eine Steuerungskonfiguration nie das gesamte Steuerungsproblem. Gemäß dieser Charakteristik stellt das Beispiel zur Interaktionsklasse 6 im Kapitel 4 eine flexible Steuerung dar. Der im Abschnitt 4.3.5 entwickelte Lösungsansatz mittels modularer Aufgabenorientierung zeigt eine komplexe Steuerungsstruktur, wobei zeitgleich nur wenige Teile der Steuerung aktiv sind. Aufgrund der strukturellen Komplexität ist die Steuerung schwer wartbar.

Dieses Phänomen erkannte Maletzki [67] bereits bei SRS. Bei MRS verschärft sich das Problem aufgrund der Interaktionen zwischen Robotern. Zur übersichtlichen Realisierung flexibler Steuerungen für SRS untersuchte Maletzki [67] erfolgreich eine Kombination der modularen Aufgabenorientierung mit dem SES/MB Framework. Dabei benutzte Maletzki nur eine Teilmenge der Beschreibungsmittel der SES. Das SES/MB Framework wurde auch zur Modellierung flexibler Problemstellungen in anderen Bereichen erfolgreich eingesetzt, wie zum Beispiel zur Modellierung von Flugszenarien in Durak et al. [33] und Karmokar et al. [15].

In Analogie zu Maletzki [67] soll in dieser Arbeit die Kombination der modularen Aufgabenorientierung mit dem SES/MB Framework zur übersichtlichen Realisierung von flexiblen Steuerungen für MRS untersucht werden. Im Gegensatz zu Maletzki [67] soll der gesamte SES/MB-Ansatz einschließlich Erweiterungen genutzt werden. Dazu werden in diesem Kapitel die Grundlagen der SES-Theorie nach Zeigler und Hammonds [134] und Erweiterungen nach Pawletta et al. [83], Folkerts et al. [36] und Schmidt [97] eingeführt. Weiterhin wird als Basis für Kapitel 6 das Konzept des Experimental Frames nach Zeigler et al. [134, 136], Rozenblit [92], Traore et al. [118] eingeführt. Als eigener Beitrag wird im Abschnitt 5.2 die prinzipielle Integration von SBC-Ansatz und SES/MB-Framework aufgezeigt.

Die Anwendung des SES/MB-Ansatzes zur Spezifikation einer flexiblen Steuerung für MRS und die Generierung ausführbarer Steuerungskonfigurationen werden in Kapitel 6 diskutiert. Am Beispiel der Interaktionsklasse 6 werden der klassische Steuerungsansatz auf Basis von Kapitel 4 und der SES/MB-unterstützte Ansatz gegenübergestellt.

5.1 System-Entity-Structure/Model-Base Framework

Die Theorie der System-Entity-Structure (SES) wurde von Zeigler [133] im Jahre 1984 eingeführt. Eine SES ermöglicht die Spezifikation von Wissen in Form eines Graphen, konkret durch einen gerichteten azyklischen Baum. Ein SES-Baum besteht aus Knoten und Kanten, welche Attribute definieren können. Ein Beispiel einer einfachen SES ist in Abbildung 5.1 gezeigt. Bei den Knoten einer SES werden zwei Typen unterschieden. Ein Knoten kann vom Typ Entität oder vom Typ beschreibender (deskriptiver) Konten sein. In der Abbildung sind Entiätenknoten zur besseren Veranschaulichung in fetter Schrift dargestellt.

Die SES-Theorie definiert drei Arten von beschreibenden Knoten: Aspekt-, Sezialisierung- und MultiAspekt-Knoten. Diese repräsentieren Beziehungen zwischen dem Vater- und den Kindknoten. Ein Aspekt-Knoten definiert eine *has-a-Beziehung* und beschreibt die Dekomposition eines Entitätenknotens in (Sub)Entitäten, sowie deren Kopplungsrelationen durch ein Attribut (couplings) am Knoten. In einer SES wird ein Aspekt-Knoten in der Regel durch das Präfix *Dec* für das englische Wort Decomposition gekennzeichnet. Ein zweiter Knotentyp, der MultiAspekt-Knoten, ist ein Spezialfall eines Aspekt-Knotens. Er hat oft das Präfix *MAsp* und besitzt eine nachfolgende Dreifachlinienkante. Der MultiAspekt-Knoten beschreibt, dass die Vater-Entität eine Komposition aus mehreren Entitätenknoten des selben Typs ist. Die Mächtigkeit der Komposition wird als Attribut des Knotens definiert, wobei auch Intervalle zulässig sind. Analog zum Aspekt-Knoten werden die Kopplungsrelationen in einem Attribut *couplings* definiert. Der dritte Typ beschreibender Knoten ist der Spezialisierungsknoten. Dieser definiert eine *is-a-Beziehung* und ist häufig durch das Präfix *Spec* gekennzeichnet. Zusätzlich kann eine Spezialisierungsbeziehung durch eine Doppelkante symbolisiert werden. Die in Abbildung 5.1 gezeigten Attribute *selection rule* und *aspect rule* sind Erweiterungen der klassischen SES-Theorie und werden später in diesem Abschnitt erläutert. Analog zu den beschreibenden Knoten können Entitätenknoten ebenfalls Attribute definieren.

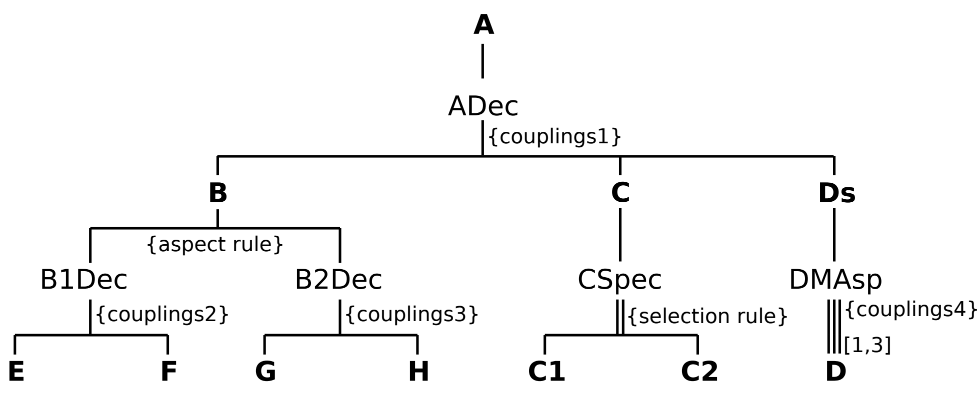


Abbildung 5.1: Beispiel einer SES

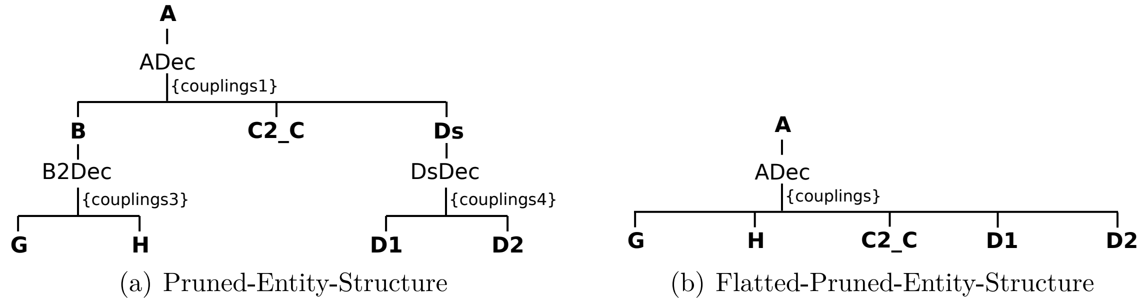
Bei näherem Betrachten der SES in Abbildung 5.1 zeigen sich strukturelle Regelmäßigkeiten. Diese folgen aus den Axiomen der SES. Die SES-Theorie definiert sechs Axiome. Tabelle 5.1 listet die Axiome mit einer Kurzbeschreibung auf. Weiterhin sind im Anhang D Beispiele zur Verletzung der Axiome aufgezeigt.

Tabelle 5.1: Axiome der SES

Axiom	Kurzbeschreibung
Alternierender Modus	Entitäten- und beschreibende Knoten müssen sich abwechseln. (Wurzel- und Blattknotten sind immer Entitätenknoten)
Gültige Brüder	Knoten mit gleichem Vater müssen unterschiedliche Namen besitzen.
Strikte Hierarchie	Alle Knoten in einem Pfad müssen unterschiedliche Namen besitzen.
Angehängte Variablen	Variablen (in der hier verwendeten Erweiterung Attribute) eines Knotens müssen unterschiedliche Namen besitzen.
Vererbung	Der Vater eines Spezialisierungsknotens erbt beim Pruning Namen, Attribute und Unterbäume eines ausgewählten Kindknotens.
Gleichförmigkeit	Knoten mit gleichen Namen in unterschiedlichen Pfaden müssen isomorphe Teilbäume spezifizieren.

Mit Hilfe der Beschreibungsmittel der SES kann eine Variantenmodellierung in Form eines Metamodells gemäß Durak et al. [33] erfolgen. In Analogie zu Durak et al. stellt die Spezifikation zulässiger Steuerungsstrukturen und deren Parametrierung ein Problem der Variantenmodellierung dar. Eine Struktur und Parametrierung bilden eine Steuerungskonfiguration. Beim SES/MB-Ansatz erfolgt in der Modellierungsphase eine strikte Trennung zwischen Konfigurationen (Systemstrukturen und Parametrierungen) und dynamischen Komponenten. Mit einer SES werden die zulässigen Konfigurationen beschrieben, während die dynamischen Komponenten, sogenannte *Basic-Models*, als wiederverwendbare Modelle in einer Modellbasis (MB) organisiert werden. Die SES spezifiziert über die Namen der Blattknoten oder Attribute der Blattknoten eine formale Verbindung zu den Basic-Models.

Um aus der Menge möglicher Konfigurationen eine auszuwählen, definiert der SES/MB-Ansatz eine Operation, welche *pruning* genannt wird. Mit der *pruning*-Operation wird der SES-Baum beschnitten. Das Ergebnis ist eine Pruned-Entity-Structure (PES). Beim *pruning* werden alle Variationspunkte einer SES aufgelöst. Die SES-Theorie definiert unterschiedliche Möglichkeiten zur Modellierung von Variantionspunkten, welche nachfolgend am Beispiel der SES in Abbildung 5.1 betrachtet werden. Eine Möglichkeit ist der Spezialisierungsknoten (CSpec). Beim *pruning* wird exakt ein Kind eines Spezialisierungsknotens ausgewählt. Die Auswahl beim *pruning* erfolgt auf Basis von Regeln, welche im Attribut *selection rule*, definiert sind. Eine weitere Möglichkeit zur Modellierung eines Variantionspunktes ist ein Entitätenknoten mit mehreren nachfolgenden Aspektknoten als Kinder (B mit B1Dec und B2Dec). Die Auswahl eines Kindes erfolgt beim *pruning* durch eine *aspect rule*. Der MultiAspekt-Knoten (DMAsp) repräsentiert ebenfalls einen Variationspunkt. Beim Pruning werden gemäß den Knotenattributen eine konkrete Anzahl zu generierender Kindknoten und die anzuwendenden Kopplungsrelationen ausgewählt. Abbildung 5.2 (a) zeigt ein Beispiel für eine PES, welche durch Auflösung der Variantspunkte der SES in Abbildung 5.1 erzeugt wurde. Die Spezifikation von Wissen mit einer SES kann gemäß der vorangegangenen Beschreibung auf verschiedene Weise erfolgen. Außerdem existieren diverse Weiterentwicklungen der SES-Theorie. In dieser Arbeit wird auf die SES-Spezifikation in Schmidt [97] aufgebaut.



(a) Pruned-Entity-Structure (b) Flatted-Pruned-Entity-Structure

Abbildung 5.2: Beispiel einer PES und einer FPES zur SES in Abbildung 5.1

Im Kontext dieser Arbeit beschreibt eine PES eine konkrete Steuerungskonfiguration. Analog zur SES definiert die PES über die Namen der Blattknoten oder Attribute der Blattknoten formale Kopplungen zu den Basic-Models in der MB. Zur Generierung eines ausführbaren Modells definiert der SES/MB-Ansatz eine *build*-Operation. Diese generiert auf Basis der Informationen in der PES und der Basic-Models ein Simulationsmodell, dass durch einen Simulator ausführbar ist. Das Teilbild SES/MB-Framework in Abbildung 5.3 zeigt schematisch das Vorgehen in Anlehnung an Schmidt [97]. Die *build*-Operation generiert 1:1 zur Struktur der PES ein modular-hierarchisches Simulationsmodell. Die Parametrierung der Basic-Models im generierten Simulationsmodell folgt aus ihrer Vorparametrierung in der MB und den Informationen in der PES. Die PES in Abbildung 5.2 spezifiziert keine Parametrierungen. In der Regel werden diese in Form von Attributen an den Blattknoten definiert. Bei der Modellierung werden aus Gründen der Übersichtlichkeit oft Hierarchieebenen eingeführt. Im Kontext eines ausführbaren Steuerungsmodells sind unnötige Hierachieebenen in der Regel unerwünscht, da sie das Laufzeitverhalten verschlechtern. Zur Eleminierung von Hierachieebenen kann vor der *build*-Operation eine Flatted-Pruned-Entity-Structure (FPES) erzeugt werden, wie in Abbildung 5.2 (b) dargestellt.

Das SES/MB-Framework nach Zeigler [131] wurde für eine interaktive Arbeitsweise entwickelt. Der Nutzer führt ein- oder mehrmals die *pruning*-Operation aus und es werden eine PES oder eine Menge von PES erzeugt. Anschließend wird die *build*-Operation ein- oder mehrmals ausgeführt und es werden die generierten Simulationsmodelle mit Hilfe eines Simulators ausgeführt. Es existiert keine Schnittstelle oder Komponente, um auf Basis der Simulationsergebnisse eine nächste *pruning*- und *build*-Operation auszuführen. In Schmidt [97] wurde das SES/MB-Framework mit der Zielstellung der Automatisierung von Experimenten erweitert. Dazu wurden neue SES-Elemente eingeführt und das Framework um neue Komponenten ergänzt. Abbildung 5.3 zeigt die Anwendung des erweiterten Frameworks, angepasst auf die Problematik flexibler Steuerungen. An dieser Stelle werden einige der Erweiterungen kurz diskutiert. Die spezielle Anwendung auf flexible Steuerungen wird im nächsten Abschnitt detailliert.

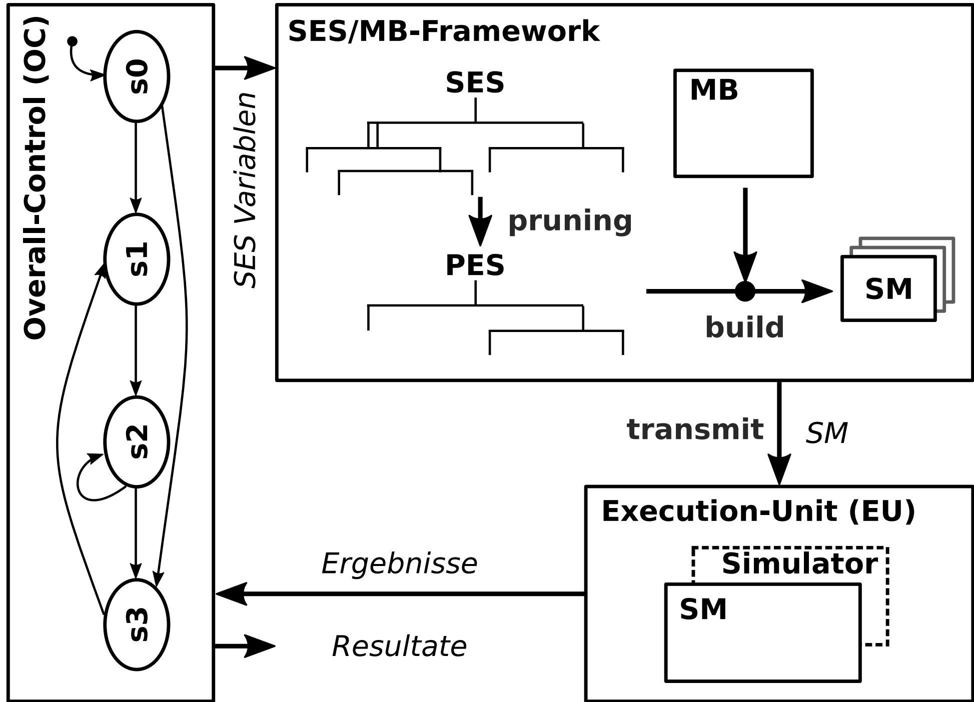


Abbildung 5.3: Erweitertes SES/MB-Framework zum automatisierten Experimentieren nach Schmidt [97], angepasst auf die Problematik flexibler Steuerungen

Eine Neuerung besteht in der Einführung globaler Variablen, welche *SES-Variablen* genannt werden. SES-Variablen dienen als Eingangsschnittstelle für das SES/MB-Framework und können zur variablen Parametrierung in SES-Attributen (couplings, selection rule, aspect rule, ...) verwendet werden. Weiterhin werden sogenannte *SES-Funktionen* eingeführt. Mit SES-Funktionen kann prozedurales Wissen in SES-Attributen beschrieben werden. SES-Funktionen können SES-Variablen als Eingangsargumente besitzen. Die Verwendung von SES-Variablen und SES-Funktionen reduziert nach Schmidt [97] oft die Komplexität eines SES-Baumes. Zulässige Wertebelegungen und Abhängigkeiten zwischen SES-Variablen können mit *Semantischen Bedingungen* beschrieben werden. Die SES-Variablen müssen vor der Ausführung einer *pruning*-Operation zulässige Wertebelegungen aufweisen. Zur Automatisierung des Experimentierens mit unterschiedlichen Systemkonfigurationen wurde das Framework in Schmidt um eine *Experiment-Control* (in Abbildung 5.3 Overall-Control (OC)) und eine Execution-Unit (EU) erweitert. Die Experiment-Control spezifiziert das auszuführende Experiment und die damit verbundenen Zielstellungen. Sie weist den SES-Variablen aktuelle Werte zu, auf dessen Basis mittels der *pruning*- und *build*-Operation ein neues SM generiert wird. Das SM wird mit der *transmit*-Operation an die EU übergeben, welche die Ausführung des SM mit einem Simulator steuert. Die Simulationsergebnisse werden von der EU an die Experiment-Control zurückgegeben. Letztere entscheidet über den weiteren Experiementverlauf. Dass heißt, ob eine neue Systemkonfiguration generiert und simuliert wird oder ob finale Ergebnisse (*overall results*) ausgegeben werden und die Abarbeitung insgesamt beendet wird.

5.2 Flexible Steuerungen auf Basis des Erweiterten SES/MB-Frameworks

Wie bereits eingeführt, besteht die Charakteristik einer flexiblen Steuerung in der situativen Anpassung an den zu steuernden Prozess unter Beachtung der vorgegebenen Zielkriterien. Der Prozesszustand wird mittels Sensoren erfasst. Auf Basis der Sensordaten können modellbasiert weitere aktuelle oder prädiktive Größen berechnet werden. Die situative Anpassung der Steuerung erfolgt durch Änderung der Steuerungskonfiguration.

Im Kontext der Arbeit erfolgt der Steuerungsentwurf und dessen Realisierung auf Basis des SBC-Ansatzes. Ein ausführbares SM wird in Verbindung mit einem echtzeitfägigen Simulator als Steuerungssoftware (Control-Software, CS) verwendet. Beim SBC-Ansatz erfolgt eine modular-hierarchische Strukturierung des SM in den drei Ebenen CM, PM und IM. Die Struktur der drei Ebenen sowie die Parametrierung der Komponenten bilden die Steuerungskonfiguration. Abbildung 5.4 zeigt konzeptionell die Anpassung einer nach dem SBC-Ansatz strukturierten Steuerung zur Laufzeit.

Die Anpassung der Konfiguration kann auf unterschiedliche Art und Weise erfolgen. Bei einem monolithischen Ansatz, wie beim Lösungsansatz für die Interaktionsklasse 6 im Abschnitt 4.3.5, sind alle Konfigurationen in einem SM und somit auch in einer CS abgebildet, wobei jeweils nur einzelne Komponenten aktiv sind. Dieser Ansatz wird im Software-Engineering als 150%-Ansatz bezeichnet [41, 44, 53]. Nach Ansicht des Autors erschwert ein monolithischer Ansatz auf Grund der Komplexität der Spezifikation das Testen und die Wartung einer Steuerung.

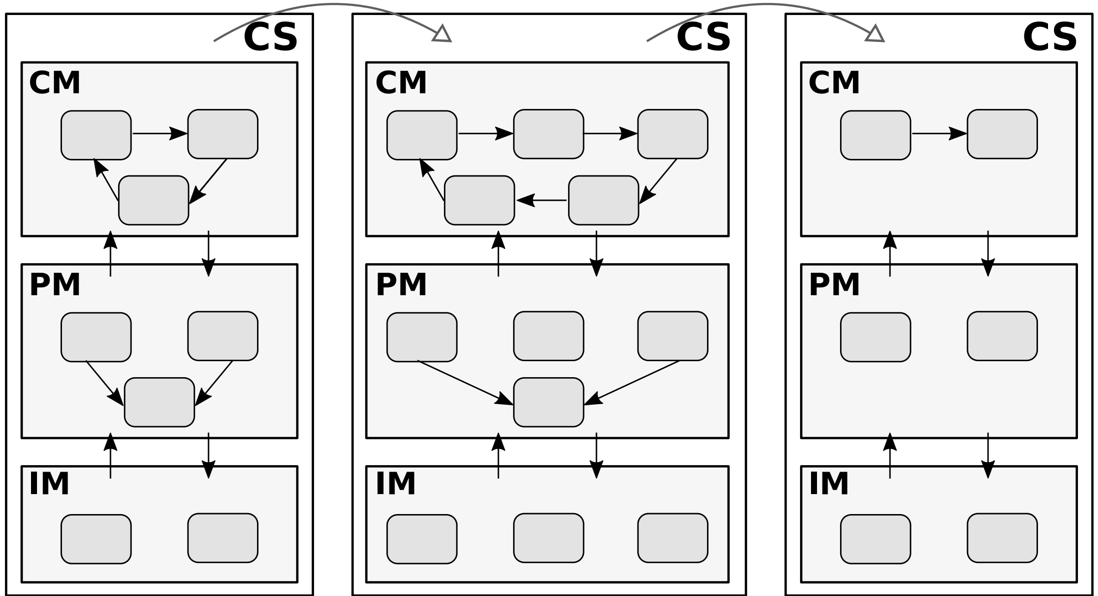


Abbildung 5.4: Änderung der Steuerungskonfiguration einer nach dem SBC-Ansatz strukturierten flexiblen Steuerung

Unter Verwendung des zuvor beschriebenen *Erweiterten SES/MB-Frameworks* können die Steuerungskonfigurationen mit einer SES spezifiziert und die Modellkomponenten der einzelnen Ebenen in einer MB organisiert werden. Die Bedingungen und Abhängigkeiten

zur Aktivierung einer bestimmten Steuerungskonfiguration können gemäß Abbildung 5.3 in der Komponente OC spezifiziert werden. Durch eine entsprechende Wertebelegung der SES-Variablen wird ein SM generiert, welches die situative Steuerungskonfiguration abbildet. Dieses SM wird durch die EU (Abb. 5.3), den echtzeitfähigen Simulator, zur Ausführung gebracht. Es ist erforderlich, dass sich in Abhängigkeit des Prozesszustandes oder geänderter Zielvorgaben, die aktuelle CS-Konfiguration selbstständig beendet. Dazu muss das SM neben der Steuerungskonfiguration den dazugehörigen Kontext, dass heißt den Gültigkeitsbereich, abbilden.

In der Simulationstheorie wurde von Zeigler [132] das Konzept des EF eingeführt, welches in Nachfolgearbeiten von Rozenblit [92], Daum und Sargent [26], Traoré und Muzy [118] sowie Schmidt [97] weiterentwickelt wurde. Ganz allgemein bildet ein EF den Kontext eines Modells ab. Abbildung 5.5 zeigt die prinzipielle Struktur eines SM unter Anwendung des EF-Konzepts in Anlehnung an Schmidt [97]. Die Struktur des EF ist nicht festgeschrieben. In dieser Arbeit bildet das Model-Under-Study (MUS) die Steuerungskonfiguration ab und der EF den Kontext zur Ausführung der aktuellen Konfiguration (Ziele, Randbedingungen, ...).

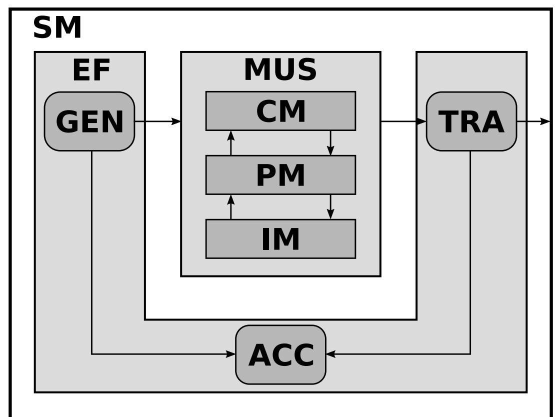


Abbildung 5.5: Struktur eines SM mit einer Steuerungskonfiguration (MUS) und dem dazugehörigen Kontext (EF)

Bereits Zeigler [132] empfiehlt den EF als ein gekoppeltes Modell, bestehend aus den Komponenten GEN, Acceptor (ACC) und TRA, umzusetzen. Dabei müssen nicht immer alle Komponenten vorhanden sein und die Kopplungsrelationen können variieren. Der GEN definiert die Einflussgrößen auf das MUS. Der TRA sammelt und berechnet relevante Untersuchungsergebnisse, sogenannte Bestimmungsgrößen, auf Basis der Ergebnisse des MUS. Der ACC bewertet die Gültigkeit der aktuellen Ausführung und beendet diese zeit- oder zustandsbasiert. Die internen Kopplungen des EF und die Kopplungen zwischen EF und MUS können je nach Untersuchungsziel variieren. Nach Beendigung der aktuellen Ausführung gibt das SM gemäß Abbildung 5.3 Ergebnisse an die EU zurück. Die EU leitet die Ergebnisse an die OC weiter. In diesem Zusammenhang ist es wichtig, dass aktuelle Zustände an die OC zurückgegeben werden, damit die OC einen gültigen Anfangszustand für die nachfolgende Steuerungskonfiguration einstellen kann.

5.3 Zusammenfassung

Es wurden die Grundlagen des SES/MB-Frameworks, bekannte Erweiterungen sowie das Konzept des EF eingeführt. Als eigener Beitrag wurde die prinzipielle Integration von SES/MB-Framework und SBC-Ansatz zur Realisierung flexibler Steuerungen für MRS betrachtet. Die Kombination beider Ansätze unterstützt die Spezifikation von Steuerungsvarianten mit einer SES. Weiterhin können unter Nutzung einer MB mit Aufgabenmodulen zur Laufzeit verschiedene Steuerungskonfigurationen generiert und ausgeführt werden. Es ist wichtig, den Kontext einer Steuerungskonfiguration zu spezifizieren. Hierzu wird auf das Konzept des EF zurückgegriffen und dessen Anwendung im Kontext dieser Arbeit erläutert. Die Anwendung und softwaretechnische Umsetzung der in diesem Kapitel eingeführten Grundlagen für MRS erfolgt im nachfolgenden Kapitel.

6 Implementierung von TOC für MRS

Ausgehend von den in der Arbeit entwickelten Interaktionsklassen wird in diesem Kapitel die softwaretechnische Umsetzung von TOC Applikationen für Multirobotersysteme (MRS) diskutiert. Die im Abschnitt 4.3 aufgezeigten Interaktionsklassen wurden in Form verschiedener Fallbeispiele unter Verwendung der MATLAB-Umgebung prototypisch implementiert. Dabei wurde auf verschiedene in der Forschungsgruppe CEA entwickelte Toolboxen zurückgegriffen, die im Rahmen der Arbeit zum Teil angepasst und erweitert wurden.

Im ersten Abschnitt wird zunächst auf die Verwendung von MATLAB als SBC-Plattform und die verwendeten Toolboxen eingegangen. Anschließend wird die softwaretechnische Umsetzung von TOC für MRS anhand von Beispielen für ausgewählte Interaktionsklassen aufgezeigt. Es erfolgt eine detaillierte Spezifikation der einzelnen Komponenten mittels des PDEVS-RCP-V2-Formalismus unter Verwendung der erweiterten DEVS-Diagramm-Notation nach Unterabschnitt 3.2.3. Die DEVS-Diagramme können unter Verwendung der nachfolgend diskutierten *DEVS Toolbox for MATLAB* nahezu 1:1 implementiert werden, wie im Anhang E gezeigt.

6.1 MATLAB als SBC-Plattform

Die softwaretechnische Umsetzung der zuvor eingeführten Konzepte erfolgt in der MATLAB-Umgebung. MATLAB wurde Ende der 1970er Jahre von Cleve Moler an der Universität New Mexico entwickelt [75]. Pawletta [80] ordnet MATLAB den wissenschaftlich-technischen Entwicklungsumgebungen (Scientific-Computing-Environment, SCE) zu. SCEs dienen der effizienten Prototypenentwicklung und sind durch zwei Kernkonzepte charakterisiert: (i) die interpretative Arbeitsweise und (ii) eine erweiterte array-orientierte Programmierung. Die interpretative Arbeitsweise ermöglicht einen kurzen Fehlerkorrekturzyklus. SCEs unterstützen als array-orientierte Sprachen eine Vielzahl an Vektor- und Matrixoperationen mit denen numerische Berechnungen sehr effizient kodiert werden können. SCEs unterstützen heute sowohl imperative als auch objektorientierte Programmiertechniken. Sie unterstützen das Erstellen von anwendungsorientierten Bibliotheken, welche als Toolboxen bezeichnet werden. Toolboxen erweitern homogen das Grundsystem einer SCE um neue Funktionalitäten.

Die Umsetzung von Multi-Robotersteuerungen in der MATLAB-Umgebung auf Basis der Konzepte dieser Arbeit, erfordert die Bereitstellung folgender Funktionalitäten in der MATLAB-Umgebung: (i) Konnektivität mit unterschiedlichen Robotertypen von verschiedenen Herstellern (eine Middleware), (ii) Unterstützung eines echtzeitfähigen

DEVS-Formalismus, (iii) Umsetzung des Erweiterten SES/MB-Frameworks.

6.1.1 Auswahl einer Roboter-Middleware

Für eine Steuerungsentwicklung nach dem SBC-Ansatz muss die MATLAB-Umgebung um Befehle zur und Kommunikation mit Robotersystemen erweitert werden. In Chinello [16] wird die *KUKA Control Toolbox* für MATLAB vorgestellt. Diese unterstützt eine interaktive Programmierung von Robotern des Herstellers KUKA in der MATLAB-Umgebung. Die Toolbox basiert auf dem Client-Server-Modell gemäß Abschnitt 2.1.2. Auf dem Controller des zu steuernden Roboters wird ein spezieller Befehlsinterpreter installiert. Der Controller fungiert als Server, welcher über eine Schnittstelle Befehle von einem MATLAB-Programm empfängt und zur Ausführung bringt. Das MATLAB-Programm fungiert als Client. Der gleiche Ansatz wird von Maletzki bei der *KUKA-KRL-Toolbox for MATLAB and Scilab* [43] verwendet. Im Gegensatz zur Toolbox von Chinello entspricht die Befehlssyntax bei Maletzki eins-zu-eins der KUKA-Robotic-Language (KRL). Somit können in der MATLAB-Umgebung interaktiv entwickelte Steuerungen bei Bedarf einfach nach KRL portiert, übersetzt und direkt auf dem Roboterkontroller installiert werden. Weiterhin wird bei Maletzki in gleicher Art und Weise die freie SCE Scilab [104] unterstützt.

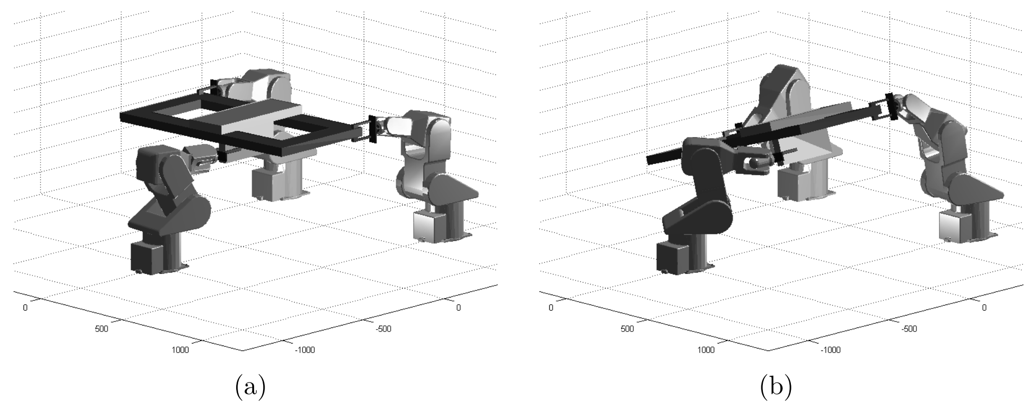
Nachteilig für Multirobotersysteme (MRS) ist, dass diese Toolboxen nur die Konnektivität zu Robotern des Herstellers KUKA unterstützen. Christern und Schmidt [98] analysierten die Befehlssätze verschiedener Gelenkarmroboter unterschiedlicher Hersteller und leiteten einen abstrahierten Befehlssatz ab, welchen Sie als Toolbox für MATLAB und für Scilab implementierten [20]. Gleichzeitig stellt die Toolbox einen Befehlsinterpreter für KUKA und KAWASAKI-Roboter zur Verfügung. Die Struktur des Befehlsinterpreters ermöglicht eine einfache Portierung auf Gelenkarmroboter anderer Hersteller.

Otto erweiterterte die Toolbox von Christern und Schmidt zu einem CAR System in der MATLAB-Umgebung [78] [77]. Die Erweiterung umfasst 3D-Modelle und Kinematikmodelle spezifischer KUKA- und KAWASAKI-Roboter. Weiterhin können über gängige CAD-Schnittstellen statische Umgebungsobjekte und zu manipulierende, dynamische Objekte in die Visualisierung integriert werden. Damit kann eine umfassende Prozessvisualisierung erfolgen, welche eine komfortable Offline-Programmierung unterstützt. Basierend auf dem Client-Server Ansatz können Befehlsinterpreter für virtuelle Roboter eingebunden werden. Damit sind wesentliche Voraussetzungen für die Steuerungsentwicklung von MRS erfüllt. Eine zu entwickelnde Steuerung kann virtuell erprobt werden und nahezu unverändert zum Steuern realer Roboter nach dem SiL Prinzip gemäß Abschnitt 2.3.2 verwendet werden. Die erweiterte Toolbox wird Robotic Control and Visualisation (RCV) Toolbox genannt. Eine kurze Übersicht ausgewählter Funktionen zeigt Tabelle 6.1.

Mit der RCV-Toolbox können verschiedene Roboter innerhalb der MATLAB-Umgebung visualisiert werden. Es lassen sich komplexe Anwendungsszenarien und das Zusammenspiel mehrerer Roboter simulieren. Demgemäß können Interaktionen zwischen Robotern eines Teams getestet werden. Ein Beispiel des Autors zur virtuellen Erprobung eines Roboterteams mit drei Robotern zeigt Abbildung 6.1 [39]. Die Roboter bewegen gemeinsam Objekte, was eine zeitliche und örtliche Abstimmung erfordert.

Tabelle 6.1: Ausgewählte Befehle der RCV-Toolbox

Befehl	Kurzbeschreibung
	Befehle zur Robotersteuerung ohne Präfix:
robot	öffnet/schließt eine Verbindung über TCP/IP oder Serial
rpoint	definiert Trajektorien durch Koordinaten und Bewegungseigenschaften
rmove	startet die Ausführung einer Bewegung durch einen Roboter
ris	überprüft den Status einer Roboterbewegung
rstop	stoppt die Ausführung einer Roboterbewegung sofort
rrun	reaktiviert die Ausführung einer Bewgung nach den Stop durch rstop
	Befehle zur Visualisierungssteuerung mit *VirtualRobot.* Präfix:
create	erstellt und platziert einen virtuellen Roboter
place_env	platziert ein nicht greifbares virtuelles Objekt
place_part	platziert ein durch den Roboter greifbares Objekt



(a) (b)

Abbildung 6.1: Beispiel zur Visualisierung mehrerer Roboter mittels RCV-Toolbox [39]

Die RCV-Toolbox wurde als Roboter-Middleware für die Implementierungen im Rahmen dieser Arbeit verwendet.

6.1.2 Umsetzung eines echtzeitfähigen DEVS-Formalismus

Die *DEVS Toolbox for MATLAB* nach Deatcu [28] integriert den DEVS-Formalismus in die MATLAB-Umgebung. Konkret unterstützt die Toolbox den PDEVS-Formalismus. Sie verfügt über eine Bibliothek mit Beispielen und Templates. Die Toolbox ist objektorientiert programmiert. Benutzerdefinierte PDEVS-Klassen werden durch Vererbung aus Toolbox-Klassen abgeleitet. Durch die homogene Umsetzung in der MATLAB-Umgebung können PDEVS-Modelle auf alle Funktionalitäten des MATLAB-Systems zurückgreifen. Dies ermöglichte eine effiziente Umsetzung des für diese Arbeit notwendigen PDEVS-RCP-V2-Formalismus nach Abschnitt 4.1.2. Mit der im vorangegangenen Abschnitt diskutierten RCV-Toolbox und der um PDEVS-RCP-V2 erweiterten DEVS Toolbox for MATLAB sind die softwaretechnischen Voraussetzungen gegeben, um Steuerungsmodelle nach dem SBC-Ansatz gemäß Abschnitt 2.3.3 in der MATLAB-Umgebung umzusetzen.

Während der Implementierungsarbeiten zeigte sich, dass für eine effiziente Steuerungsentwicklung die Debugging-Optionen der DEVS Toolbox for MATLAB unzureichend sind. In van Mierlo, Tendeloo und Vangheluwe [120] wird darauf verwiesen, dass umfangreiche Debugmodi eine Schlüsseltechnologie zur Entwicklung sicherer Steuerungen sind. Im Rahmen dieser Arbeit wurden neue Funktionalitäten zur Plausibilitätsprüfung von Modellen entwickelt. Es wurden: (i) ein graphischer Debug-Modus, (ii) eine automatische Prüfung von Kopplungsrelationen und (iii) eine erweiterte Unterstützung des Entwicklers beim Erstellen neuer Modelle umgesetzt. Abbildung 6.2 zeigt ein Beispiel für das grafisch unterstützte Debuggen einer Robotersteuerung. Der Entwickler kann relevante Zustände und Ereignisse eines PDEVS-Modells zur Laufzeit analysieren.

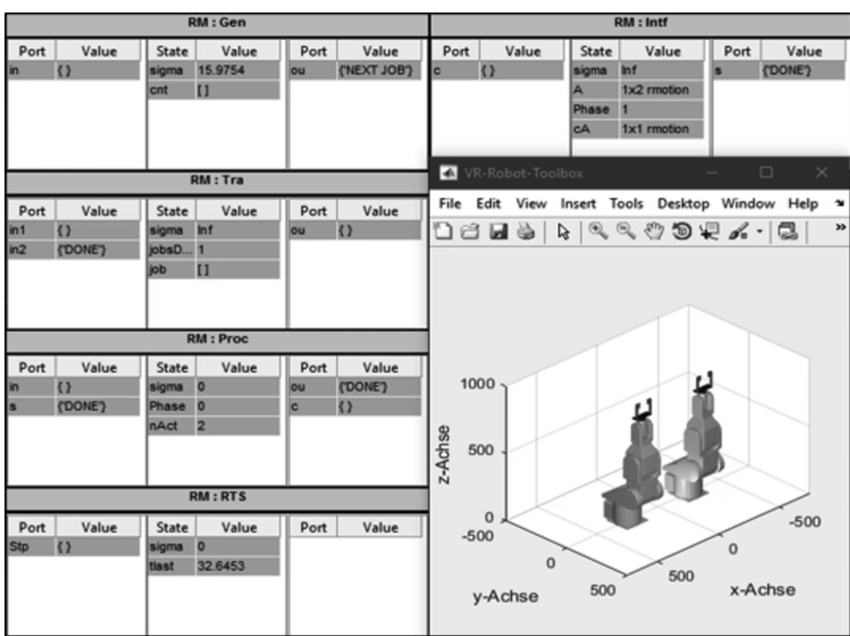


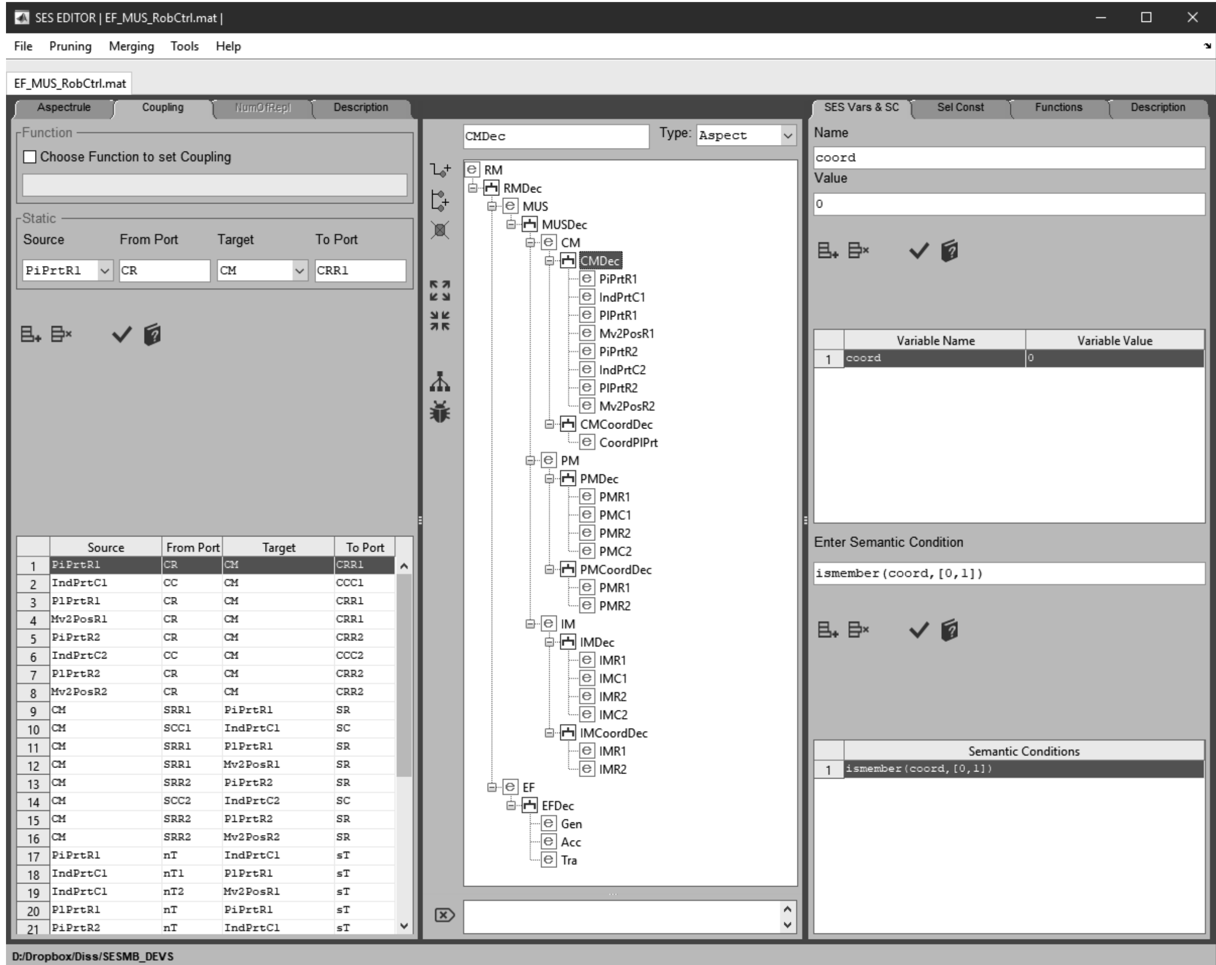
Abbildung 6.2: Grafisch unterstütztes Debuggen einer PDEVS-basierten Robotersteuerung

In Deatcu et al. [27] und Freymann et al. [38] wurden bereits die Entwicklung von Steuerungen für Robotersysteme unterschiedlicher Hersteller unter Verwendung der DEVS Toolbox for MATLAB und der RCV-Toolbox aufgezeigt. Die PDEVS-RCP-V2-Erweiterung vereinfacht die dort aufgezeigte Steuerungsentwicklung.

6.1.3 Umsetzung des SES/MB-Ansatzes

Die *SES Toolbox for MATLAB/Simulink* (SES/MB Tbx) implementiert die Funktionalitäten des SES/MB-Ansatzes nach Kapitel 5, ohne die Komponenten OC und EU des Erweiterten SES/MB-Frameworks in Abbildung 5.3. Die Toolbox ist eine Eigenentwicklung der Forschungsgruppe CEA [82], an welcher der Autor beteiligt war. Die SES/MB Tbx bietet eine graphische Benutzeroberfläche zur intuitiven Spezifikation einer SES. Fehlerhafte Nutzereingaben werden durch automatisches Prüfen der zugrundeliegenden SES-Axiome verhindert. Abbildung 6.3 zeigt den sogenannten SES-Editor, welcher die graphische Nutzerschnittstelle zum Erstellen einer SES darstellt. Die Oberfläche ist in drei Bereiche gegliedert. Diese unterstützen den Nutzer bei der systematischen Erstellung einer SES. Im mittleren Fenster wird der SES-Baum editiert und in einer Dateibrowser-View

dargestellt. Die anderen Bereiche der Benutzeroberfläche umfassen die Parametrierung eines ausgewählten Knotens und die Definition von globalen Eigenschaften der SES.



	Source	From Port	Target	To Port
1	PiPrtR1	CR	CM	CRR1
2	IndPrtC1	CC	CM	CCC1
3	PlPrtR1	CR	CM	CRR1
4	Mv2PosR1	CR	CM	CRR1
5	PiPrtR2	CR	CM	CRR2
6	IndPrtC2	CC	CM	CCC2
7	PlPrtR2	CR	CM	CRR2
8	Mv2PosR2	CR	CM	CRR2
9	CM	SRR1	PiPrtR1	SR
10	CM	SCC1	IndPrtC1	SC
11	CM	SRR1	PlPrtR1	SR
12	CM	SRR1	Mv2PosR1	SR
13	CM	SRR2	PiPrtR2	SR
14	CM	SCC2	IndPrtC2	SC
15	CM	SRR2	PlPrtR2	SR
16	CM	SRR2	Mv2PosR2	SR
17	PiPrtR1	nT	IndPrtC1	sT
18	IndPrtC1	nT1	PlPrtR1	sT
19	IndPrtC1	nT2	Mv2PosR1	sT
20	PlPrtR1	nT	PiPrtR1	sT
21	PiPrtR2	nT	IndPrtC1	sT

Abbildung 6.3: Graphische Nutzerschnittstelle der SES Toolbox for MATLAB/Simulink

Die Toolbox unterstützt vielfältige Erweiterungen der klassischen SES-Theorie . Wesentliche Erweiterungen sind nach [84] die Einführung von SES-Variablen, von SES-Funktionen und des charakteristischen Attributs *mb*. Wie im Kapitel 5 erläutert, definieren die SES-Variablen eine Input-Schnittstelle der SES und gestatten eine variable Parametrierung von SES-Attributen (Auswahlregeln, Kopplungsbeziehungen oder Entitätenparameter). SES-Funktionen ermöglichen eine prozedurale Parametrierung von SES-Attributen. Häufig finden SES-Funktionen ihre Anwendung um variable Kopplungsrelationen zu beschreiben. Das charakteristische Attribut *mb* defniert eine Verbindung zwischen Entitäten-Knoten und Komponenten in einer Modellbasis. Die Auswertung des Attributs *mb* erfolgt durch die im Kapitel 5 beschriebene *build*-Methode. Die SES/MB Toolbox unterstützt eine Modellgenerierung für verschiedene Simulatoren. Derzeit umfasst die Toolbox Modellgenerartoren für Simulink, Dymola und OpenModelica. Beispiele zur Anwendung der Toolbox werden in Schmidt et al. [99], Pawletta et al. [83] und Pawletta, Pascheka und Schmidt [81] gezeigt. Im Rahmen dieser Arbeit wurde ein Modellgenerator für die DEVS Toolbox for MATLAB entwickelt sowie die Benutzerfreundlichkeit und Software-Zuverlässigkeit der SES/MB Toolbox verbessert.

6.1.4 Umsetzung des Erweiterten SES/MB-Frameworks

Im Rahmen dieser Arbeit wurde das Erweiterte SES/MB-Framework nach Abschnitt 5.2 unter Verwendung der zuvor vorgestellten MATLAB-Toolboxen umgesetzt. In Abbildung 6.4 ist die prinzipielle softwaretechnische Umsetzung gezeigt. Die wesentlichen Verknüpfungen der einzelnen Komponenten sind in der Abbildung durch Nummern gekennzeichnet.

In der Entwurfs- und Automatisierungsphase werden mittels der DEVS Toolbox for MATLAB, kurz DEVS Tbx, Basic-Models entwickelt und in einer MB organisiert (Nummer 0b). Die möglichen Steuerungskonfigurationen werden in einer SES unter Verwendung des graphischen Editors der SES Toolbox spezifiziert (Nummer 0a). Wie bereits ausgeführt, wird über das *mb-Attribut* eine formale Kopplung zwischen Entitäten der SES und den Basic-Models der MB spezifiziert.

Die Generierung und Ausführung einer Steuerungskonfiguration wird durch die Komponente OC gemanagt. Die OC definiert die Steuerungsziele und Randbedingungen. Die OC wird im einfachsten Fall als ein MATLAB-Skript implementiert. Die Generierung einer Steuerungskonfiguration startet mit der Wertebelegung von SES-Variablen in der OC. Anschließend ruft die OC die *pruning*-Operation der SES Toolbox auf (Nummer 1) und übergibt die SES-Variablen als Eingangsargumente. Als Ergebnis wird eine PES zurückgeliefert. Danach wird die *build*-Methode der SES Toolbox aufgerufen, welche mit dem in dieser Arbeit entwickelten DEVS-Modellgenerator aus der PES und den Basic-Models ein Simulationsmodell (SM) generiert (Nummer 1). Im Ergebnis der Modellgenerierung wird das SM an die OC zurückgeliefert (Nummer 2). Das SM kodiert eine konkrete Steuerungskonfiguration und ist gemäß den Ausführungen in Abschnitt 5.2 strukturiert.

Anschließend setzt die OC die notwendigen Konfigurationsparameter für die EU und veranlasst einen Simulationslauf des SM unter Steuerung der EU (Nummer 3). Das heißt, die aktuelle Steuerungskonfiguration wird ausgeführt (Nummer 4). Die Prozessinteraktion (Nummer 5) erfolgt auf Basis des in der Arbeit entwickelten PDEVS-RCP-V2-Formalismus nach Abschnitt 4.1.2 und unter Nutzung der RCV Toolbox als Roboter-Middleware.

Bei flexiblen Steuerungen gemäß Kapitel 5 besitzt eine Steuerungskonfiguration eine begrenzte Gültigkeit und kann sich selbst terminieren. Bei der Terminierung werden relevante Ergebnisse gesammelt und von der EU an die OC zurückgegeben (Nummer 6). Die OC kann basierend auf den Ergebnissen eine neue Steuerungskonfiguration generieren oder sich beenden. Die gewonnenen Resultate werden ausgegeben (Nummer 7).

In der Entwurfsphase beziehungsweise beim Übergang in die Automatisierungsphase kann mittels der RCV Toolbox eine Visualisierung der zu steuernden Prozessumgebung erstellt werden und mittels des gesamten Frameworks eine Offline-Entwicklung erfolgen. Je nach Anwendungsfall sind auch Kombinationen aus realen und virtuellen Prozesskomponenten umsetzbar, wodurch eine sukzessive Inbetriebnahme erfolgen kann.

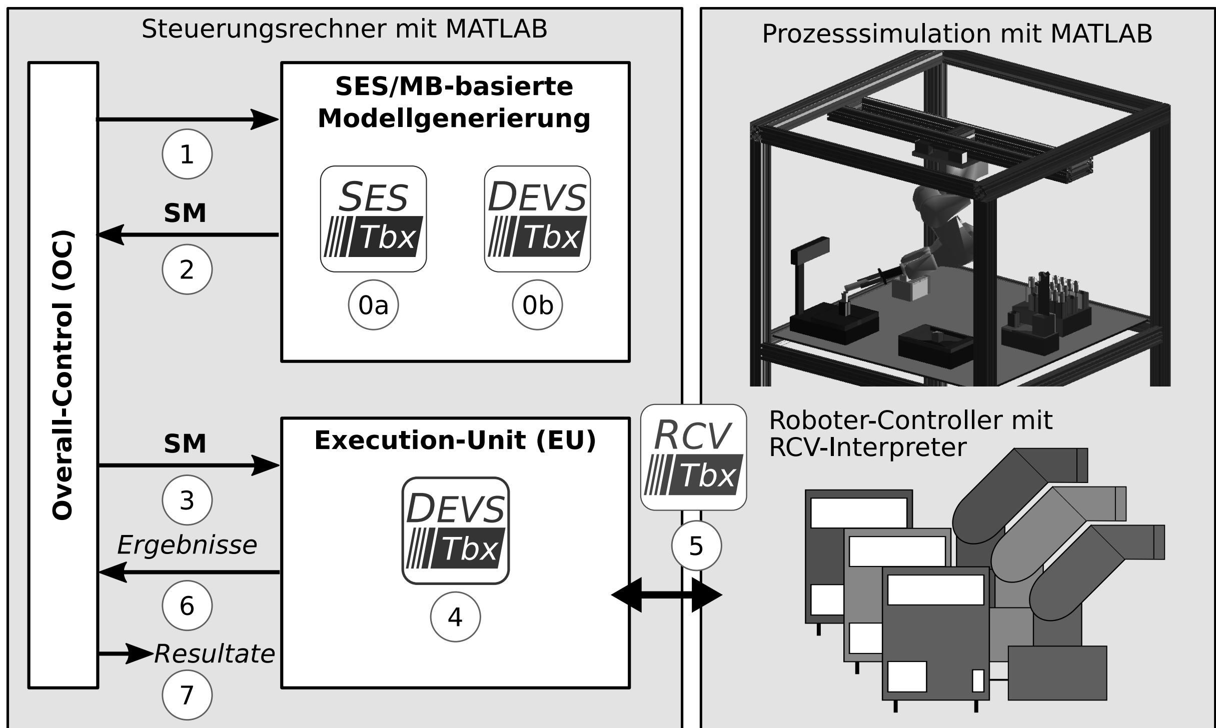


Abbildung 6.4: Softwaretechnische Umsetzung des Erweiterten SES/MB-Frameworks zur Realisierung flexibler Multi-Robotersteuerungen

Nachfolgend wird die Umsetzung von Interaktionen in einem MRS durch Aufgaben im Sinne einer TOC untersucht. Zunächst erfolgt die Umsetzung ohne Verwendung des Erweiterten SES/MB-Frameworks. Anschließend wird an einem Beispiel die Anwendung des Erweiterten SES/MB-Frameworks gezeigt und dem klassischen Vorgehen gegenübergestellt.

6.2 Umsetzung ausgewählter Lösungsansätze

In diesem Abschnitt wird die detaillierte Spezifikation von Steuerungskomponenten für ausgewählte Interaktionsklassen gezeigt. Die Spezifikationen basieren auf den in Abschnitt 4.3 entwickelten konzeptionellen Lösungsansätzen. Zu Beginn erfolgt eine Konkretisierung des Lösungsansatzes der Interaktionsklassen 1 und 2. Danach erfolgt eine Konkretisierung des Steuerungsansatzes für Interaktionsklasse 3, welche die Komponente Roboterteam einführt. Analog zur Interaktionsklasse 3 können die konzeptionell ähnlichen Ansätze der Interaktionsklassen 4 und 5 in Abschnitt 4.3 umgesetzt werden. Aus diesem Grund werden letztere hier nicht separat betrachtet. Den höchsten Schwierigkeitsgrad weist Interaktionsklasse 6 auf. Sie beinhaltet alle Interaktionsprinzipien der eingeführten Interaktionsklassen. Die Umsetzung des konzeptionellen Lösungsansatzes der Interaktionsklasse 6 nach Abschnitt 4.3 wird detailliert diskutiert.

Am Beispiel der Komponenten Roboter und Roboterteam wird gezeigt, dass Modellkomponenten schrittweise weiterentwickelt werden können. Die detaillierte Spezifikation der Modellkomponenten erfolgt mit erweiterten DEVS-Diagrammen. Im Anhang E ist der zugehörige MATLAB/DEVS-Code aufgelistet.

6.2.1 Interaktionsklasse 1 und 2

Eine grundlegende Eigenschaft der Interaktionsklasse 1 und 2 (Abb. 2.20) besteht in der Trennung der Arbeitsräume der Roboter. Wie in Abbildung 4.13 gezeigt, können die Roboter unabhängig voneinander arbeiten. Es kann für jeden Roboter die gleiche Aufgabensequenz definiert werden:

$\mathrm{Pos} = \mathrm{IdPrt}(\mathrm{IB}, \mathrm{A\ oder\ B}) \rightarrow \mathrm{Move}(\mathrm{Pos}) \rightarrow \mathrm{PickPrt} \rightarrow \mathrm{Move}(\mathrm{OB}) \rightarrow \mathrm{PlacePrt} \rightarrow ...$

Die Aufgaben sind teilweise parametriert. Die Aufgabe IdPrt (identify part) definiert einen Rückgabewert Pos (position). Dieser ist Eingangsparameter der nachfolgenden Aufgabe Move. Tabelle 6.2 zeigt, dass alle Aufgaben durch eine Aufgaben-ID (task-identifier, tid) repräsentiert werden. Weiterhin sind die möglichen Parametrierungen der Aufgaben aufgelistet. Die Aufgabentransformation erfolgt durch die jeweilige Roboterkomponente (R1 oder R2) im PM und deren Interfacekomponente im IM (vgl. Abb. 4.13).

Tabelle 6.2: Kodierung und Parameter der Aufgaben für Interaktionsklasse 1 und 2

Aufgabe	**Kodierung**	**Parameter**	**Transformation**
PickPrt	tid= 1	$p1, p2 = \varnothing$	R1 oder R2
PlacePrt	tid= 2	$p1, p2 = \varnothing$	R1 oder R2
IdPrt	tid= 3	$p1 \in \{IB\}, p2 \in \{‘A‘, ‘B‘\}, p2 = \varnothing$	R1 oder R2
Move	tid= 4	$p1 \in \{‘Pos‘, ‘OB‘\}, p2 = \varnothing$	R1 oder R2

Abbildung 6.5 zeigt die Spezifikation der Aufgabe IdPrt. Die Aufgabe definiert die zwei Hauptzustände *Passive* und *Active*. Der Initialzustand ist Passive. Durch ein externes Ereignis 'next' am Eingangsport BEG (begin) erfolgt ein Wechsel in den Zustand Active. Hierbei wird zugleich das nächste interne Ereignis mit $\sigma = 0$ eingeplant. Dies führt zum Versenden des Ereignisses $(3, p1, p2)$ am Ausgangsport CMD (comand). Die Bezeichner $p1, p2$ sind Platzhalter zur Parametrierung der Aufgabe. Der Wert 3 kodiert die Aufgaben-ID tid der Aufgabe IdPrt. Nach dem Versenden des Ausgangsereignisses wird mit $\sigma = \infty$ kein weiteres internes Ereignis geplant. Der Zustand Active kann ausschließlich über ein externes Ereignis am Eingangsport STS (status) verlassen werden. Bei Eintreten des externen Ereignisses wird der Ereigniswert auf der Zustandsvariablen res (result) gespeichert und mit $\sigma = 0$ sofort ein internes Ereignis eingeplant.

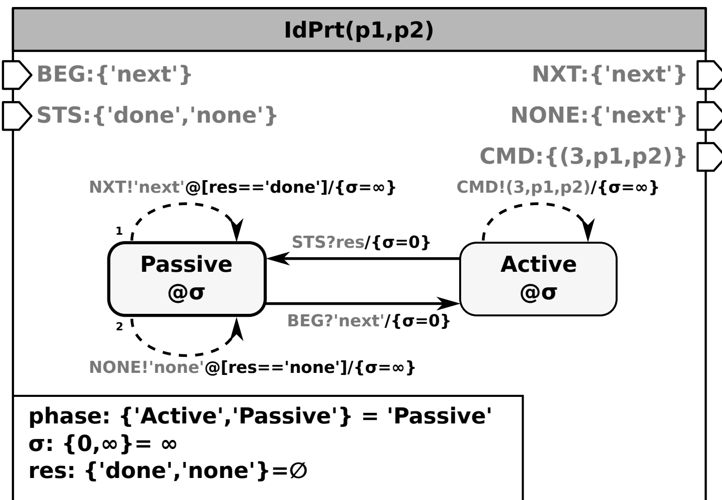


Abbildung 6.5: Erweitertes DEVS-Diagramm der Aufgabe IdPrt

Im Zustand Passive wird bei einem internen Ereignis über eine Fallunterscheidung der auf res gespeicherte Ereigniswert geprüft. Ist res == ‘done‘ wird am Ausgangsport NXT ein Ausgangsereignis ‘next‘ generiert. Wenn res == ‘none‘ gilt, konnte das gewünschte Bauteil nicht identifiziert werden. In diesem Fall wird am Ausgangsport NONE das Ereignis ‘next‘ erzeugt. Nach dem Generieren des jeweiligen Ausgangsereignisses wird mit $\sigma = \infty$ kein neues internes Ereignis eingeplant. Die Aufgabe befindet sich wieder im Initialzustand.

Das Ergebnis der Aufgabe IdPrt ist die Position eines identifizierten Bauteils. Die Position kann zur Parametrierung anderer Aufgaben genutzt werden, wie am Beispiel der Aufgabe Move(Pos) gezeigt. Im Anwendungsbeispiel erfolgt die Speicherung der ermittelten Position in der jeweiligen Roboterkomponente (R1 oder R2) auf PM-Ebene. Die Details dazu werden am Ende dieses Abschnittes bei der Umsetzung der Komponente Roboter diskutiert.

Abbildung 6.6 zeigt die Spezifikation der Aufgabe Move. Diese ähnelt der Aufgabe IdPrt. Die Aufgabe Move definiert einen Parameter p1 und wird durch die Aufgaben-ID tid=4 kodiert. Die Bezeichner der Ein- und Ausgangsports sind analog zur Aufgabe IdPrt gewählt.

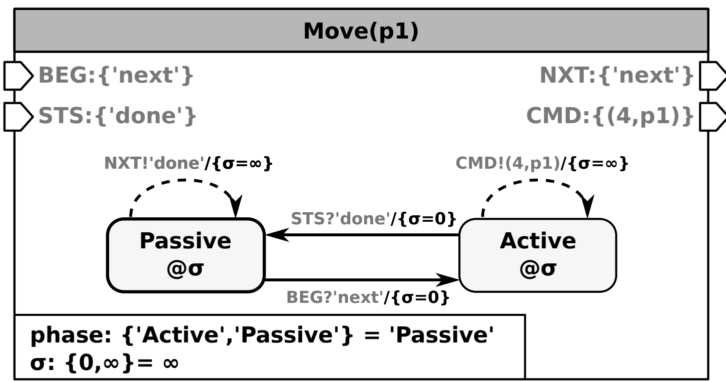


Abbildung 6.6: Erweitertes DEVS-Diagramm der Aufgabe Move

Die Aufgaben PickPrt und PlacePrt sind analog zur Spezifikation der Aufgabe Move

umsetzbar. Demzufolge kann für diese Aufgaben eine allgemeine Aufgabe definiert werden. Abbildung 6.7 zeigt die Struktur der verallgemeinerten Aufgabe *Task*.

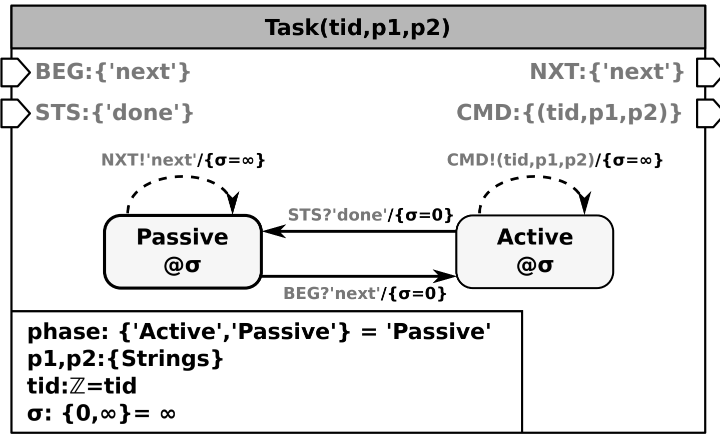


Abbildung 6.7: Erweitertes DEVS-Diagramm einer allgemeinen Aufgabe

Unter Verwendung der in Tabelle 6.2 angegebenen Aufgaben-IDs (tid) und Parametrierungen können mit der allgemeinen Aufgabe Task die drei spezifischen Aufgaben Move, PickPrt und PlacePrt realisiert werden.

Alle bisherigen Aufgaben werden mit dem Zustand Passive initialisiert. Um die Aufgabenausführung zu starten, wird eine weitere Aufgabe NoOp (no operation) definiert, welche in Abbildung 6.8 spezifiziert ist. Im konkreten Anwendungsbeispiel initiiert die Aufgabe NoOp die Ausführung der Aufgabe IdPrt für den jeweiligen Roboter. Dies erfolgt über eine Kopplung des Ausgangsports NXT mit dem Eingangsport BEG der Aufgabe IdPrt.

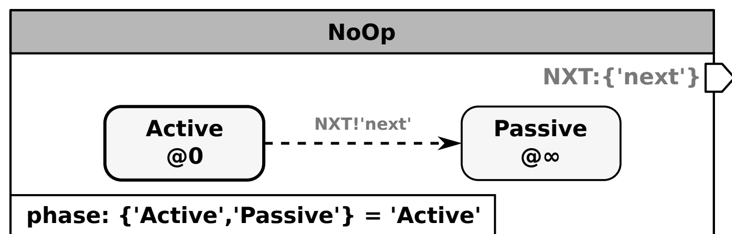


Abbildung 6.8: Erweitertes DEVS-Diagramm der Aufgabe no operation NoOp

Nachfolgend wird die Aufgabentransformation durch die jeweilige Roboterkomponente R und deren Interfacekomponente INTF gezeigt. Gemäß der Diskussion zur Implementierung komplexer Komponenten auf PM-Ebene in Abschnitt 4.3.3 sind R und INTF nach dem Multi-Modellansatz umgesetzt. Die Spezifikation von R ist in Abbildung 6.9 dargestellt. Sie definiert die drei Hauptzustände *Passive*, *Error* und *Active*, mit dem Initialzustand Passive. Soll die Komponente R eine Aufgabe ausführen, muss sich diese im Zustand Passive befinden. Die auszuführende Aufgabe wird über den Eingangsport CM_CMD als ein externes Ereignis empfangen. Der Ereigniswert entscheidet, welche Aufgabentransformation durchgeführt wird.

Ist die Aufgabe IdPrt auszuführen, verbleibt der Roboter im Zustand Passive. Im Anwendungsbeispiel ist die Positionsbestimmung stark vereinfacht. Sie wird über eine

LookUpTable realisiert. Diese definiert eine Liste mit vordefinierten Positionen der einzelnen Bauteile. Über die Parameter p1 und p2 wird die Position eines Bauteils ermittelt. Der Parameter p1 kodiert den Ort und p2 die Bauteilart. Wird eine Bauteilposition ermittelt, wird der Eintrag in der Tabelle gelöscht und der Zustand *position* mit neuen Werten belegt. Gleichzeitig wird die Zustandsvariable sts='done' gesetzt. Kann keine Bauteilposition ermittelt werden, wird sts='none' gesetzt. Zeitgleich wird mit σ=0 ein internes Ereignis eingeplant. Dies führt dazu, dass über den Ausgangsport CM_STS ein Ereignis mit dem Wert der Zustandsvariablen *sts* an das Control-Model (CM) versendet wird.

Die Ausführung der Aufgaben PickPrt, PlacePrt und Move führt immer zu einem Wechsel in den Zustand Active. Hierbei wird mit σ=0 ein internes Ereignis eingeplant. Dies führt zum Versenden eines externen Ereignisses über den Port IM_CMD an das Interfacemodell (IM). Der Ereigniswert ergibt sich aus der Zustandsvariablen *cmd* (comand) und unterscheidet sich je nach auszuführender Aufgabe. Speziell hervorgehoben sei die Aufgabe Move. Wird über den Parameter p1 der String 'Pos' empfangen, wird die zuvor im Zustand *position* gespeicherte Ortsinformation auf *cmd* gespeichert. Anderenfalls erfolgt die Bestimmung des aktuellen Wertes in *cmd* unter Verwendung einer zweiten LookUpTable. Nach dem Versenden eines Ausgangsereignisses über den Port IM_CMD an das IM, wird mit $\sigma=\infty$ kein weiteres internes Ereignis eingeplant. Der Zustand Active kann nur über ein externes Ereignis am Port IM_STS wieder verlassen werden. Dieses repräsentiert eine Statusmeldung vom IM. Sollte mit einem externen Ereigniswert IM_STS='error' ein Fehler gemeldet werden, wird der Zustand *Error* betreten. An dieser Stelle kann eine Fehlerbehandlungsroutine realisiert werden. Andernfalls (IM_STS='done') erfolgt ein Wechsel in den Zustand Passive. Weiterhin wird der Zustand sts='done' gesetzt und mit σ=0 ein internes Ereignis eingeplant. Dies führt zum Versenden eines externen Ereignisses (CM_STS!sts) an das CM. Danach wird mit $\sigma=\infty$ kein weiteres internes Ereignis eingeplant. Die Komponente R befindet sich wieder im Initialzustand und kann erneut eine Aufgabe ausführen.

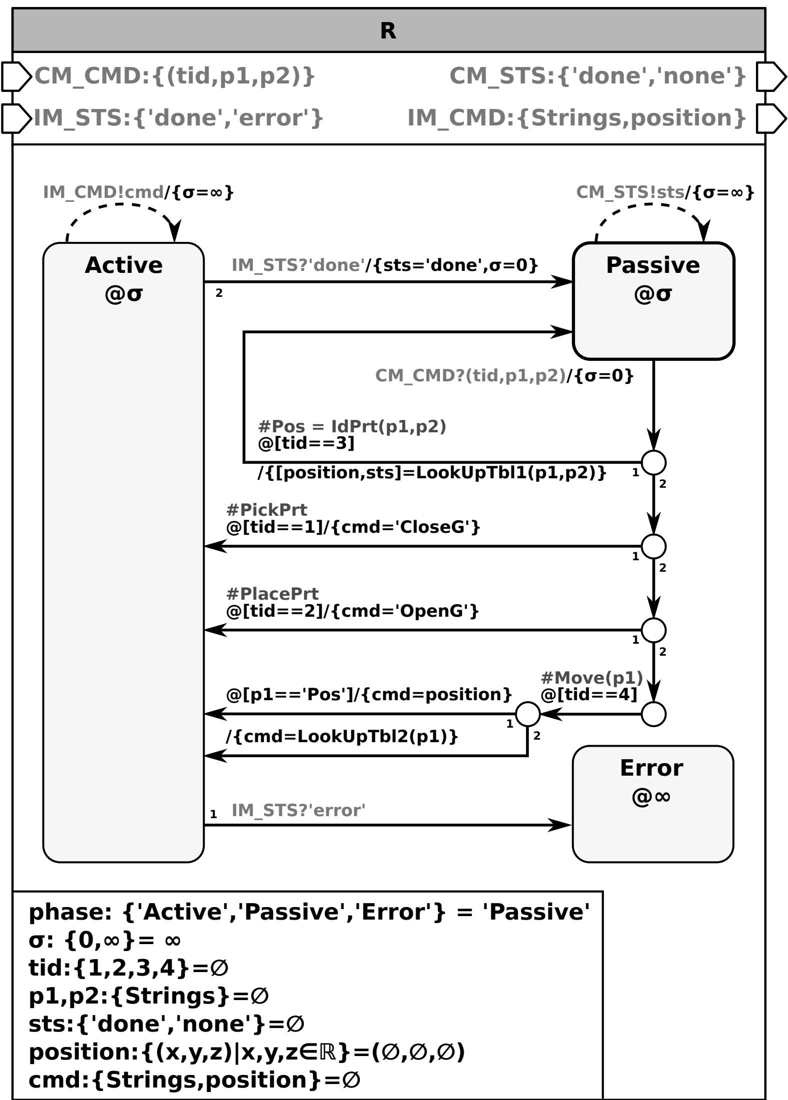


Abbildung 6.9: Erweitertes DEVS-Diagramm der Roboterkomponente R

Abbildung 6.10 zeigt das dynamische Verhalten der zu R zugehörigen Interfacekomponente INTF auf IM-Ebene. INTF empfängt von R Kommandos als Eingangsereignisse, führt Aktivitäten aus und sendet Statusmeldungen als Ausgangsereignisse zurück. Die Aktivität *rmove* startet eine Roboter- oder Greiferbewegung und gibt eine id zur Identifikation der aktuellen Bewegung zurück. Auf Basis der id ermittelt die Aktivität *ris*, ob die Bewegung abgeschlossen ist. Mit Hilfe der Aktivität *getWCT* wird die Ausführungszeit einer Bewegung überwacht. Beim Start der Bewegung, d.h. beim Zustandswechsel von Passive zu Active, wird die Wall-Clock-Time (WCT) auf die Zustandsvariable t_{CMD} geschrieben. Beim Beispiel wird angenommen, dass eine Bewegung mindestens 5 Sekunden (σ=5) benötigt und nach maximal 10 Sekunden (getWCT>t_{CMD}+10) abgeschlossen ist. Bei Überschreitung der oberen Zeitgrenze erfolgt ein Wechsel in den Zustand Error sowie das Versenden eines Ausgangsereignisses 'error'. Die Abtastzeit ist im Beispiel mit 0,1 Sekunden (σ=0.1) festgelegt. Bei Einhaltung des vorgegebenen Zeitintervalls wird ein Ausgangsereignis 'done' versendet und in den Zustand Passive gewechselt.

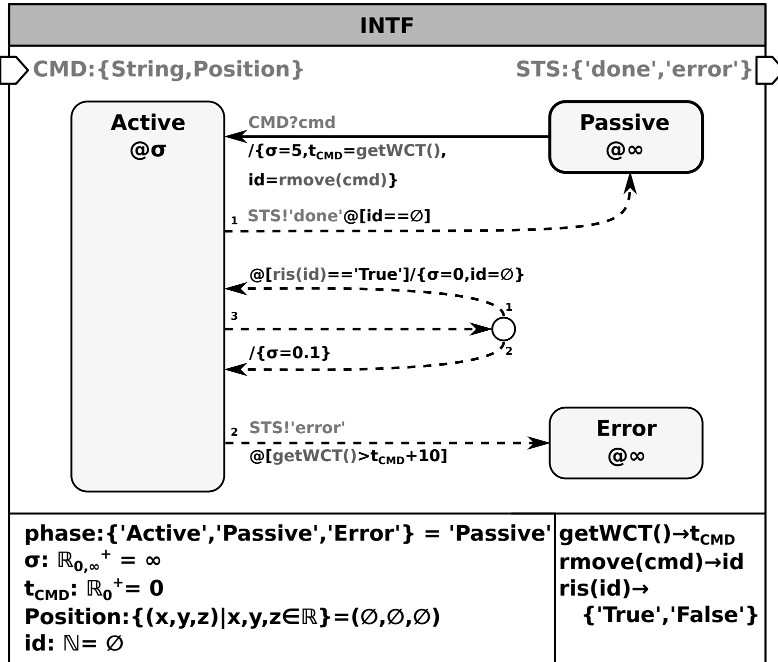


Abbildung 6.10: Erweitertes DEVS-Diagramm der Interfacekomponente INTF

Um mit einem realen Prozess zu interagieren, muss das SM auf IM-Ebene gemäß Unterabschnitt 4.1.2 um eine RTC nach Abbildung 4.3 ergänzt werden. Diese ist analog zur RTC aus Teilbild 4.4 (c) definiert und synchronisiert die Virtual-Time (VT) und die WCT.

6.2.2 Interaktionsklasse 3

Die Interaktionsklasse 3 definiert, dass sich die Roboter in einem gemeinsamen Arbeitsraum bewegen. Folglich können, im Gegensatz zu den Interaktionsklassen 1 und 2, Kollisionen auftreten. Im Anwendungsbeispiel in Abbildung 2.20 ist der Zugriff auf den Input-Buffer (IB) kritisch und sollte exklusiv erfolgen. Die Roboter müssen dementsprechend ihren Zugriff auf den IB koordinieren. Zur Lösung des Problems wird, wie in Abbildung 4.15 gezeigt, das Prinzip des wechselseitigen Ausschlusses (Mutex) genutzt. Dieses wird durch die Einführung der zwei Aufgaben *Lock* und *UnLock* modelliert, welche den Zugriff auf gemeinsame Ressourcen koordinieren. In Abbildung 4.15 ist für beide Roboter die gleiche Aufgabenfolge definiert:

$\text{Pos} = \text{IdPrt(IB, A oder B)} \rightarrow \text{Lock(IB)} \rightarrow \text{Move(Pos)} \rightarrow \text{PickPrt} \rightarrow \text{Move(SPos)} \rightarrow$
$\text{UnLock(IB)} \rightarrow \text{Move(OB)} \rightarrow \text{PlacePrt} \rightarrow \text{Move(SPos)} \rightarrow ...$

Tabelle 6.3 zeigt die Kodierung und Parametrierung der Aufgaben für das Anwendungsbeispiel. Die Umsetzung der Aufgabe *IdPrt* wurde im vorherigen Unterabschnitt in Abbildung 6.5 vorgestellt. Alle anderen Aufgaben sind gemäß der in Abbildung 6.7 eingeführten allgemeinen Aufgabe *Task* umsetzbar. Zur Aufgabentransformation ist eine Modifikation der zuvor entwickelten Komponente R sowie die Einführung einer neuen Komponente RT (Roboterteam) erforderlich.

Tabelle 6.3: Parameter und Kodierung der Aufgaben für Interaktionsklasse 3

Aufgabe	Kodierung	Parameter	Transformation
PickPrt	tid= 1	$p1, p2 = \varnothing$	R1 oder R2
PlacePrt	tid= 2	$p1, p2 = \varnothing$	R1 oder R2
IdPrt	tid= 3	$p1 \in \{‘IB‘\}, p2 \in \{‘A‘, ‘B‘\}, p2 = \varnothing$	R1 oder R2
Move	tid= 4	$p1 \in \{‘Pos‘, ‘IB‘, ‘OB‘, ‘SPos‘\}, p2 = \varnothing$	R1 oder R2
Lock	tid=-1	$p1 \in \{‘IB‘\}, p2 = \varnothing$	R1 oder R2
UnLock	tid=-2	$p1 \in \{‘IB‘\}, p2 = \varnothing$	R1 oder R2

Abbildung 6.11 zeigt das dynamische Verhalten der erweiterten Komponente R. Diese spezifiziert, im Gegensatz zu Abbildung 6.9, eine zusätzliche Kommunikation mit der neuen Komponente RT.

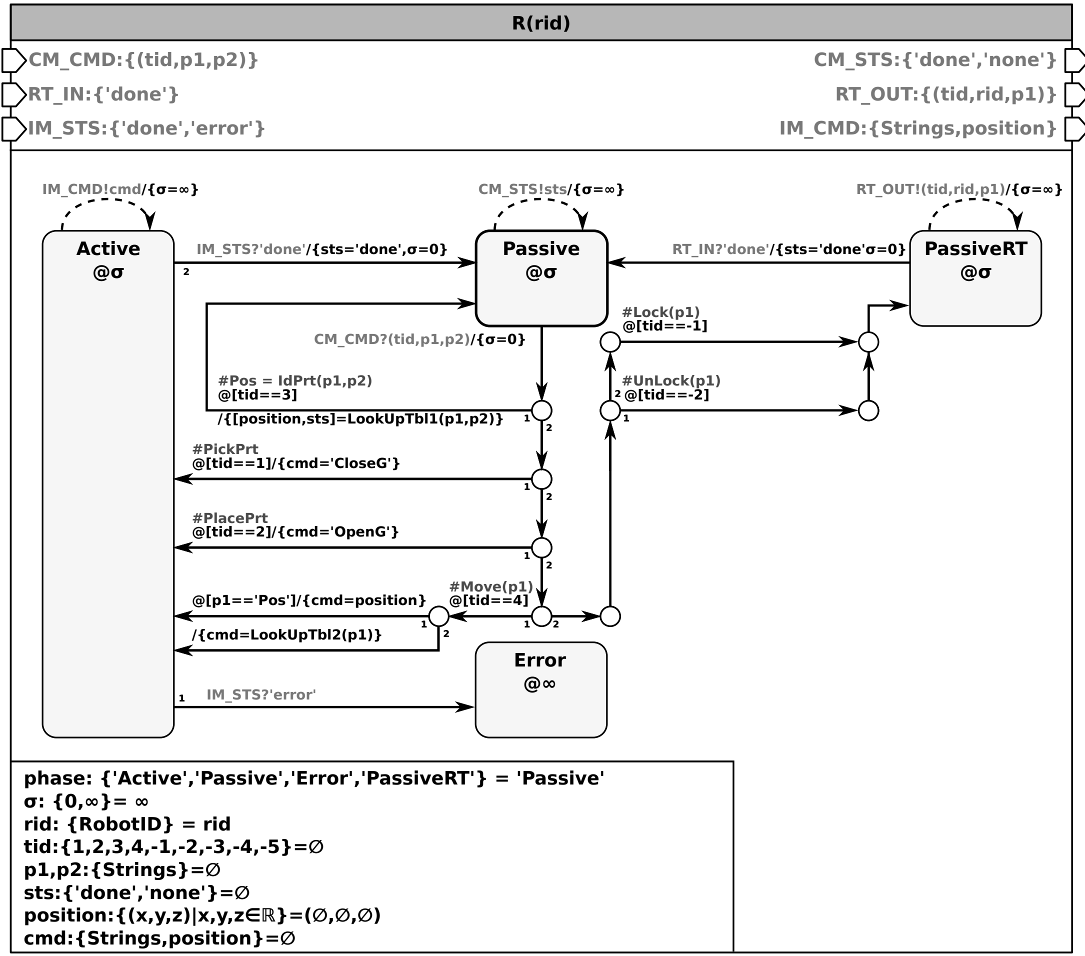


Abbildung 6.11: Erweitertes DEVS-Diagramm der Roboterkomponente R

Hierfür wird ein neuer Eingangsport RT_IN und ein neuer Ausgangsport RT_OUT definiert. Weiterhin wird ein neuer Zustand PassiveRT eingeführt. Die Aufgabentransformation der neu eingeführten Aufgaben Lock und UnLock führt zu einem Wechsel in den Zustand PassiveRT. Gleichzeitig wird aufgrund von σ=0 ein Ausgangsereignis über den Port RT_OUT an die Komponente RT gesendet. Das Ausgangsereignis beinhaltet die Aufgaben-ID *tid*, die Roboteridentifikation *rid* sowie einen Parameter p1. Das RT

(Abb. 6.12) verarbeitet das Ereignis und antwortet dem Roboter R mit einem 'done'-Ereignis am Eingangsport RT_IN, wenn die Aufgabe ausgeführt werden kann. Bei Empfang des 'done'-Ereignisses, wechselt R in den den Zustand Passive und versendet gleichzeitig (σ=0) ein Ausgangsereignis 'done' über den Port CM_STS an das Control-Model (CM).

Das dynamische Verhalten der Komponente RT ist in Abbildung 6.12 dargestellt. Der strukturelle Aufbau folgt dem Multi-Modellansatz.

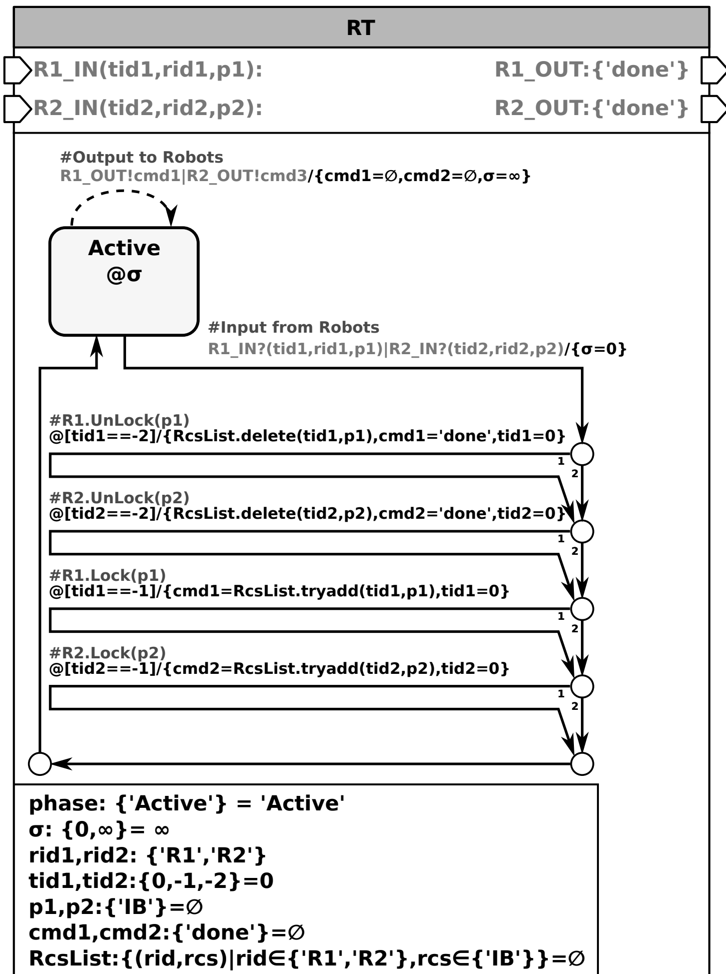


Abbildung 6.12: Erweitertes DEVS-Diagramm der Komponente Roboterteam RT

Das RT koordiniert im Anwendungsbeispiel den Zugriff der Roboter auf gemeinsame Ressourcen, im Beispiel auf den IB. Hierfür definiert das RT jeweils einen Ein- und Ausgangsport zur Kommunikation mit den Roboterkomponenten. Die Roboter können die Reservierung einer Ressource zeitgleich beauftragen. In diesem Fall erhält der Roboter R1 den Vorzug, da sein Reservierungswunsch zuerst überprüft wird. Die Reservierung erfolgt mittels eines Ressourcenbezeichners. Die Information, welcher Roboter welche Ressource reserviert, wird in einer Liste *RcsList* (resource list) hinterlegt. Eine Ressource kann der Liste nur hinzugefügt werden, wenn diese nicht bereits in der Liste vorkommt. Eine erfolgreiche Reservierung wird den Roboterkomponenten über ein 'done'-Ereignis mitgeteilt. Wird eine Ressource wieder freigegeben, so kann ihr Besitz unmittelbar an einen anderen Roboter übergehen. Hieraus folgt, dass aufgrund der Nebenläufigkeit zuerst die Aufgabe UnLock auszuwerten ist. Erst danach erfolgt die Auswertung der Aufgabe

Lock.

Das Zusammenwirken der Aufgaben Lock und UnLock der CM-Ebene mit den Roboterkomponenten R1, R2 und der Komponente Roboterteam RT der PM-Ebene ist komplex. Beide Roboter können im Anwendungsbeispiel die Ressource IB für sich reservieren. Die Reservierung erfolgt durch Ausführung der Aufgabe Lock(IB), durch welche ein Ausgangsereignis (tid,rid,p) an RT gesendet wird. RT speichert alle empfangenen Informationen auf roboterbezogenen Zustandsvariablen, im Beispiel *tid1, tid2, p1, p2, rid1, rid2*. Die Auswertung der Eingangsereignisse erfolgt ebenfalls roboterbezogen. Beantragt beispielsweise der Roboter R1 den Zugriff auf IB (Lock(IB)), so empfängt das RT das Ereignis R1_IN?(tid1,rid1,p1). In diesem Fall sind die Zustandsvariablen mit den Werten tid1=-1, rid1='R1', p1='IB' belegt. Die Auswertung des externen Ereignisses ergibt, dass der Roboter R1 den Zugriff auf IB beantragt. Demgemäß wird das Tupel ('R1','IB') in die Ressourcenliste RcsList eingetragen, wenn IB nicht bereits durch den Roboter R2 benutzt wird. Bei erfolgreicher Reservierung erhält die Roboterkomponente R1 ein 'done'-Ereignis und kann die Ausführung der Aufgabe Lock(IB) beenden. Sollte R2 nebenläufig die Aufgabe Lock(IB) ausführen wollen, so erhält dieser vom RT solange kein 'done'-Ereignis, bis die Ressource IB durch R1 freigegeben wird (UnLock(IB)).

6.2.3 Interaktionsklasse 6

Die Interaktionsklasse 6 beinhaltet ebenfalls das Agieren mehrerer Roboter in einem gemeinsamen Arbeitsraum. Sie müssen: (i) Kollisionen vermeiden, (ii) ihre Bewegungsabläufe koordinieren und (iii) gemeinsam Aufgaben lösen. In Unterabschnitt 4.3.5 wurden zwei Lösungsansätze für das Referenzbeispiel zur Interaktionsklasse 6 entwickelt. Nachfolgend wird der in Abbildung 4.24 gezeigte Lösungsansatz mit drei Robotern konkretisiert. Es werden die in Tabelle 6.4 aufgelisteten Aufgaben verwendet. Diese können wie bei Interaktionsklasse 3 beschrieben, mit Ausnahme der Aufgabe *IdPrt*, gemäß der Struktur der allgemeinen Aufgabe *Task* (Abb. 6.7) umgesetzt werden. Weiterhin wurde im Lösungsansatz in Abbildung 4.24 auf der CM-Ebene die spezielle Komponente RT eingeführt, welche die komponierte Aufgabe *PickPlaceSync* (Abb. 4.20)spezifiziert. Sie beschreibt die temporäre Bildung eines Roboterteams zur gemeinsamen Ausführung einer Aufgabensequenz, wie in diesem Fall den gemeinsamen Bauteiltransport. Ausgehend von den zuvor eingeführten Komponenten R und RT der PM-Ebene, werden nachfolgend die wesentlichen Erweiterungen beschrieben, die für die Transformation der neuen Aufgaben notwendig sind.

Tabelle 6.4: Parameter und Kodierung der Aufgaben für Interaktionsklasse 6

Aufgabe	**Kodierung**	**Parameter**	**Transformation**
PickPrt	tid= 1	$p1, p2 = \varnothing$	R1, R2 oder RT
PlacePrt	tid= 2	$p1, p2 = \varnothing$	R1, R2 oder RT
IdPrt	tid= 3	$p1 \in \{\text{‘IB‘}\}, p2 \in \{\text{‘A‘}, \text{‘B‘}, \text{‘C‘}\}, p2 = \varnothing$	R1 oder R2
Move	tid= 4	$p1 \in \{\text{‘Pos‘}, \text{‘IB‘}, \text{‘OB‘}, \text{‘SPos‘}\}, p2 = \varnothing$	R1, R2 oder RT
Lock	tid=-1	$p1 \in \{\text{‘IB‘}\}, p2 = \varnothing$	R1 oder R2
UnLock	tid=-2	$p1 \in \{\text{‘IB‘}\}, p2 = \varnothing$	R1 oder R2
JoinRT	tid=-3	$p1, p2 = \varnothing$	R1 oder R2
DefineRT	tid=-4	$p1, p2 \in \{\text{‘R1‘}, \text{‘R2‘}\}$	RT
Wait4RT	tid=-5	$p1, p2 = \varnothing$	RT
ExitRT	tid=-6	$p1, p2 \in \{\text{‘R1‘}, \text{‘R2‘}\}$	RT
Sleep	tid=-7	$p1, p2 = \varnothing$	R1 oder R3
WakeUp	tid=-8	$p1 \in \{\text{‘R1‘}, \text{‘R3‘}\}, p2 = \varnothing$	R1 oder R3

Abbildung 6.13 zeigt das erweiterte DEVS-Diagramm der Roboterkomponente R, welches ausgehend von der Spezifikation in Abbildung 6.11 entwickelt wurde. Analog zur vorherigen Umsetzung kann R eine neue Aufgaben transformieren, wenn R sich im Zustand Passive (Initialzustand) befindet. Die vom Zustand Passive ausgehende Fallunterscheidung ist im rechten Teil entsprechend den neu hinzugekommenen Aufgaben erweitert. Zusätzlich wurde der Zustand *ActiveRT* eingeführt und die Wertemengen der Eingangsereignisse angepasst. Aufgrund von $\sigma = \infty$ wartet R im Zustand Passive auf ein externes Ereignis vom CM am Eingangsport CM_STS.

Die Transformation der Team-bezogenen Aufgaben führt zu einem Zustandswechsel nach PassiveRT mit $\sigma = 0$. Das interne Ereignis führt zu einem Ausgangsereignis an die Komponente RT auf PM-Ebene, wie im vorangegangenen Abschnitt bei R für Interaktionsklasse 3 beschrieben. Dann wartet R ($\sigma = \infty$) auf ein externes Ereignis vom RT am Eingangsport RT_IN, welches ein auszuführendes Kommando (Aktivität) enthält. Bei Empfang wird in den Zustand ActiveRT gewechselt und ein internes Ereignis ($\sigma = 0$) eingeplant. Dieses bewirkt ein Ausgangsereignis IM_CMD mit der aktuell auszuführenden Aktivität an die zugehörige Interfacekomponente INTF (Abb. 6.10). Aufgrund von $\sigma = \infty$ verbleibt R für die Zeitdauer der Kommandoausführung im Zustand ActiveRT. Im Fehlerfall erhält R im Zustand ActiveRT von INTF ein Eingangsereignis 'error' und wechselt in den Zustand Error. Hier kann eine entsprechende Fehlerbehandlung erfolgen. Bei erfolgreicher Kommandoausführung erhält R im Zustand ActiveRT von INTF am Eingangsport IM_STS ein 'done' Ereignis. Es wird in den Zustand PassiveRT gewechselt und mit $\sigma = 0$ ein internes Ereignis eingeplant, woraufhin ein Ausgangsereignis über den Port RT_OUT an RT gesendet wird. Dieses informiert RT über die erfolgreiche Kommandoausführung. Aufgrund von $\sigma = \infty$ wird im Zustand PassiveRT auf ein externes 'done' Ereignis vom RT am Port RT_IN gewartet, welches zum Wechsel in den Zustand Passive (den Initialzustand) führt und aufgrund von σ=0 ein Ausgangsereignis 'done' über den Port CM_STS an das CM sendet. Mit $\sigma = \infty$ verbleibt R im Zustand Passive bis über ein externes Ereignis eine neue Aufgabentransformation ausgelöst wird.

Eine Besonderheit stellt die Transformation der Team-bezogenen Aufgabe *Sleep* dar. Hier wird im Gegensatz zu allen anderen Aufgaben beim Wechsel in den Zustand PassiveRT mit $\sigma = \infty$ kein internes Ereignis eingeplant und somit kein Ausgangsereignis an RT gesendet.

Das "Aufwecken" von R kann nur durch Ausführung der Aufgabe *WakeUp* durch einen anderen Roboter R des Teams erfolgen. Auf das Zusammenspiel der Aufgaben Sleep und WakeUp wird am Ende des Abschnittes nochmals separat eingegangen.

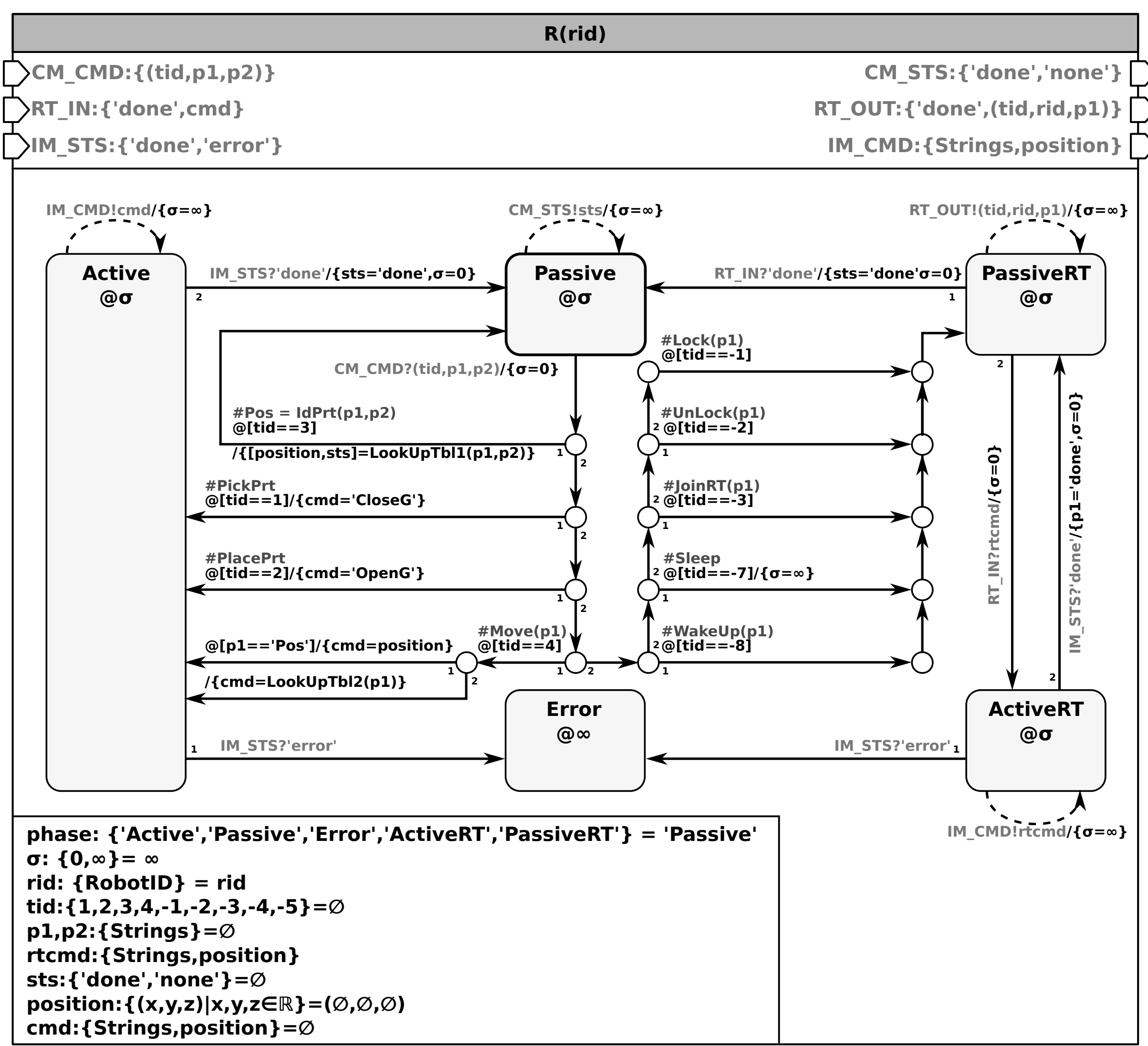


Abbildung 6.13: Erweitertes DEVS-Diagramm der Komponente Roboter R

Die auf Abbildung 6.12 basierende Spezifikation der Komponente RT ist in Abbildung 6.14 dargestellt. RT transformiert die Team-bezogenen Aufgaben. Neu sind die zusätzlichen Ein- und Ausgangsports (CM_CMD, CM_STS) zur direkten Kommunikation mit dem CM sowie die Ports zur Kommunikation mit einem dritten Roboter auf PM-Ebene. Wie zuvor definiert RT nur den Zustand Active. Bei externen Ereignissen an den Eingangsports werden zunächst die aktuellen Ereigniswerte auf entsprechende Zustandsvariablen gespeichert und mit $\sigma = 0$ ein internes Ereignis eingeplant. Es sei darauf hingewiesen, dass zeitgleich an verschiedenen Ports ein Eingangsereignis auftreten kann, wenn unterschiedliche Roboterkomponenten R oder das CM simultan ein Ereignis an das RT übermitteln. Die Auswertung per Fallunterscheidung erfolgt gemäß der Logik, wie sie bereits im Unterabschnitt 6.2.2 bei der Interaktionsklasse 3 diskutiert wurde. Der Zugriff auf die Interfacekomponente INTF (Abb. 6.10) eines Roboters erfolgt nicht direkt durch die Komponente RT, sondern über die R Komponente des jeweiligen Roboters. Die

entsprechende Kommunikation wurde zuvor bei R im Kontext des Zustandes ActiveRT beschrieben.

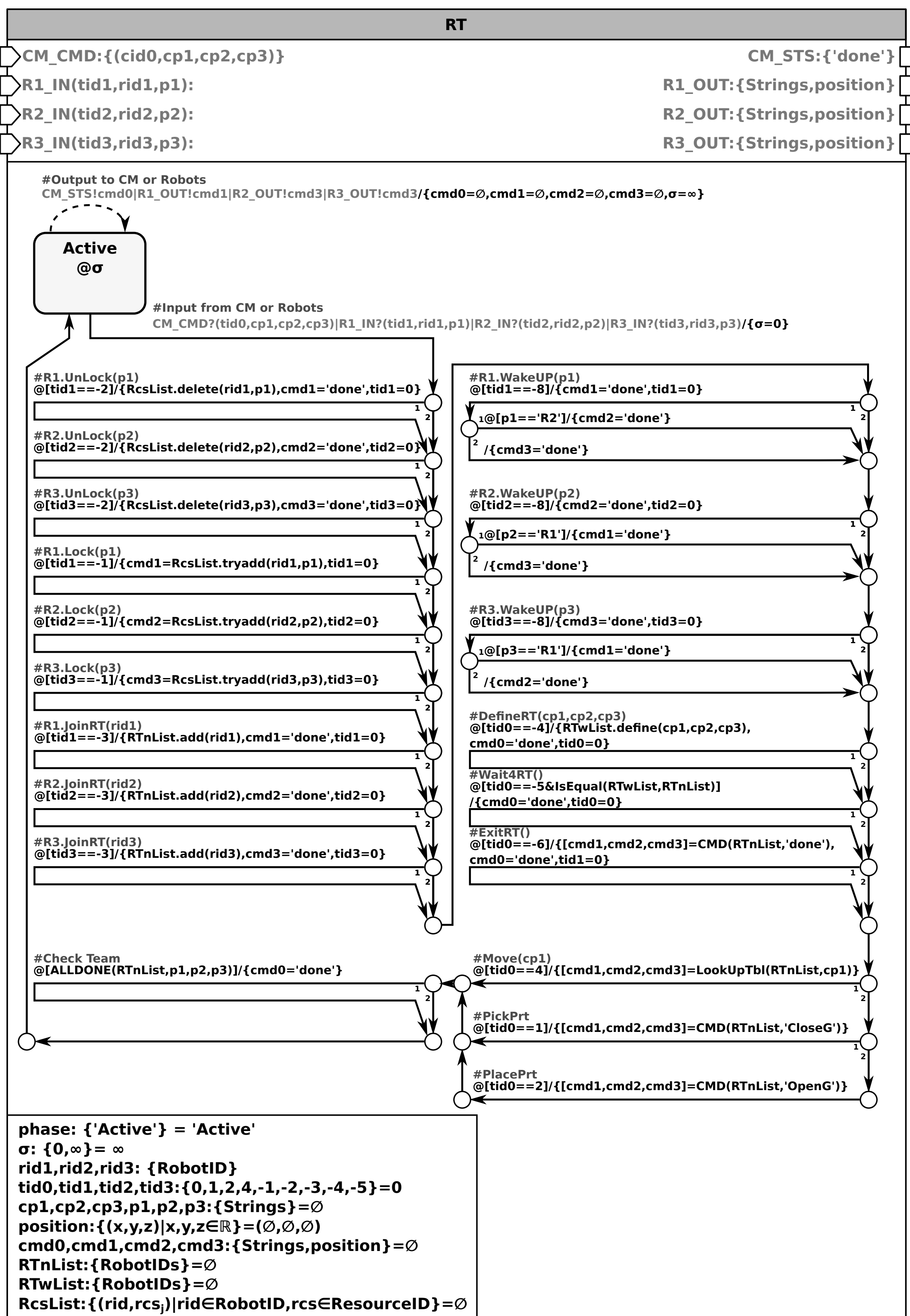


Abbildung 6.14: Erweitertes DEVS-Diagramm der Komponente Roboterteam RT

In Abbildung 6.15 wird das Zusammenspiel der Aufgaben *Sleep* und *WakeUp* am Beispiel der Roboterkomponeten R1 und R3 sowie der Roboterteamkomponente RT gezeigt. Bei Ausführung der Aufgabe Sleep wechselt die Roboterkomponente R1 in den Zustand PassiveRT. Wie bereits zuvor beschrieben wird bei der Aufgabe Sleep kein Ausgangserereignis an das RT gesendet und R1 wartet ($\sigma = \infty$) im Zustand PassiveRT, wodurch R1 keine weiteren Aufgaben ausführen kann. Die Reaktivierung von R1 erfolgt durch die Roboterkomponente R3, wenn diese die Aufgabe *WakeUp* ausführt. Durch WakeUp wird von R3 ein Ausgangsereignis mit entsprechenden Ereigniswerten an die Komponente RT gesendet. RT wertet die übermittelten Ereigniswerte aus und verschickt jeweils ein Ausgangsereignis 'done' an die Komponenten R1 und R3. Dadurch wechseln die Roboterkomponenten in den Ausgangszustand *Passive* und signalisierten dem CM, dass ihre aktuelle Aufgabe abgeschlossen wurde.

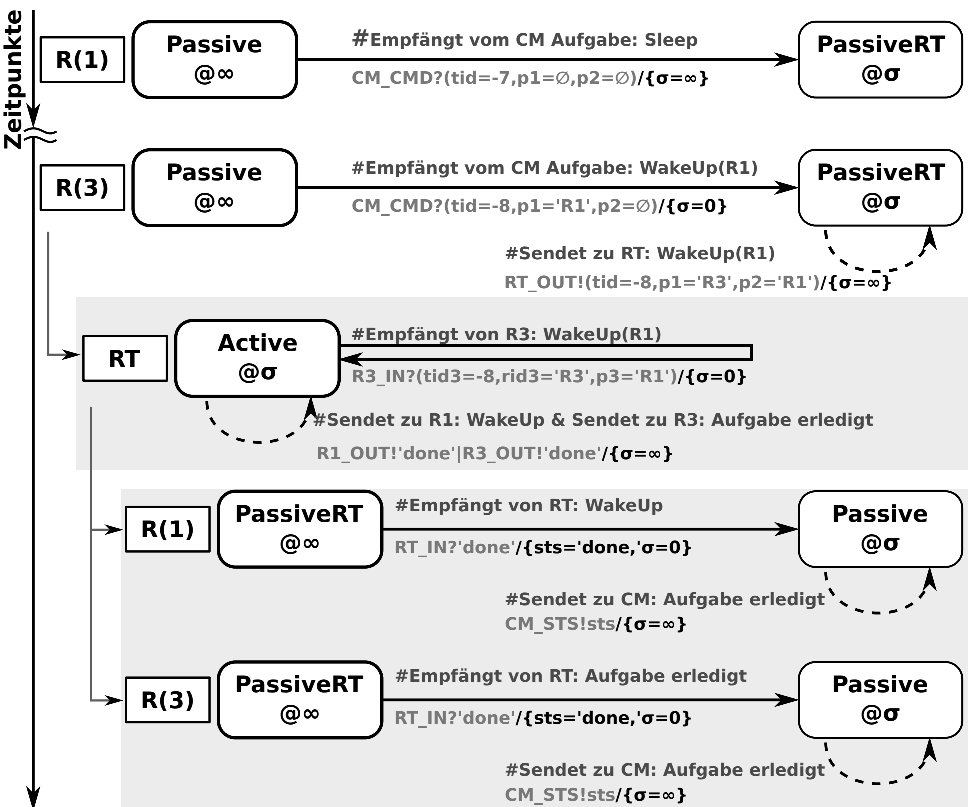


Abbildung 6.15: Zusammenspiel der Aufgaben Sleep und WakeUp mit Roboter- und Roboterteam-Komponenten

Die DEVS-Diagramme zum Referenzbeispiel der Interaktionsklasse 6 sind relativ komplex. Der Implementierungs-, Test- und Wartungsaufwand der Komponenten steigt stark an. Gemäß den Ausführungen im Abschnitt 5.2 entspricht die Entwicklung nach dem Multi-Modellansatz einer 150%-Modellierung.

6.2.4 Interaktionsklasse 6 mit Erweitertem SES/MB-Framework

Wie bereits in der Zusammenfassung im Abschnitt 4.4 angemerkt, stellt die Spezifikation und Implementierung flexibler Steuerungen eine Herausforderung für Entwickler dar.

Das Beispiel zur Interaktionsklasse 6 weist mit der temporären Integration eines dritten Roboters typische Eigenschaften einer flexiblen Steuerung auf. Als ein alternativer Modellierungsansatz wurde im Kapitel 5 die Kombination des SBC-Ansatzes mit dem System-Entity-Structure/Model-Base (SES/MB) Framework diskutiert und im Abschnitt 6.1.4 die notwendige softwaretechnische Basis vorgestellt. Nachfolgend wird eine Lösungsvariante für das Referenzbeispiel unter Nutzung des SES/MB-Frameworks diskutiert.

Abbildung 6.16 zeigt eine SES, die zwei Steuerungskonfigurationen spezifiziert.

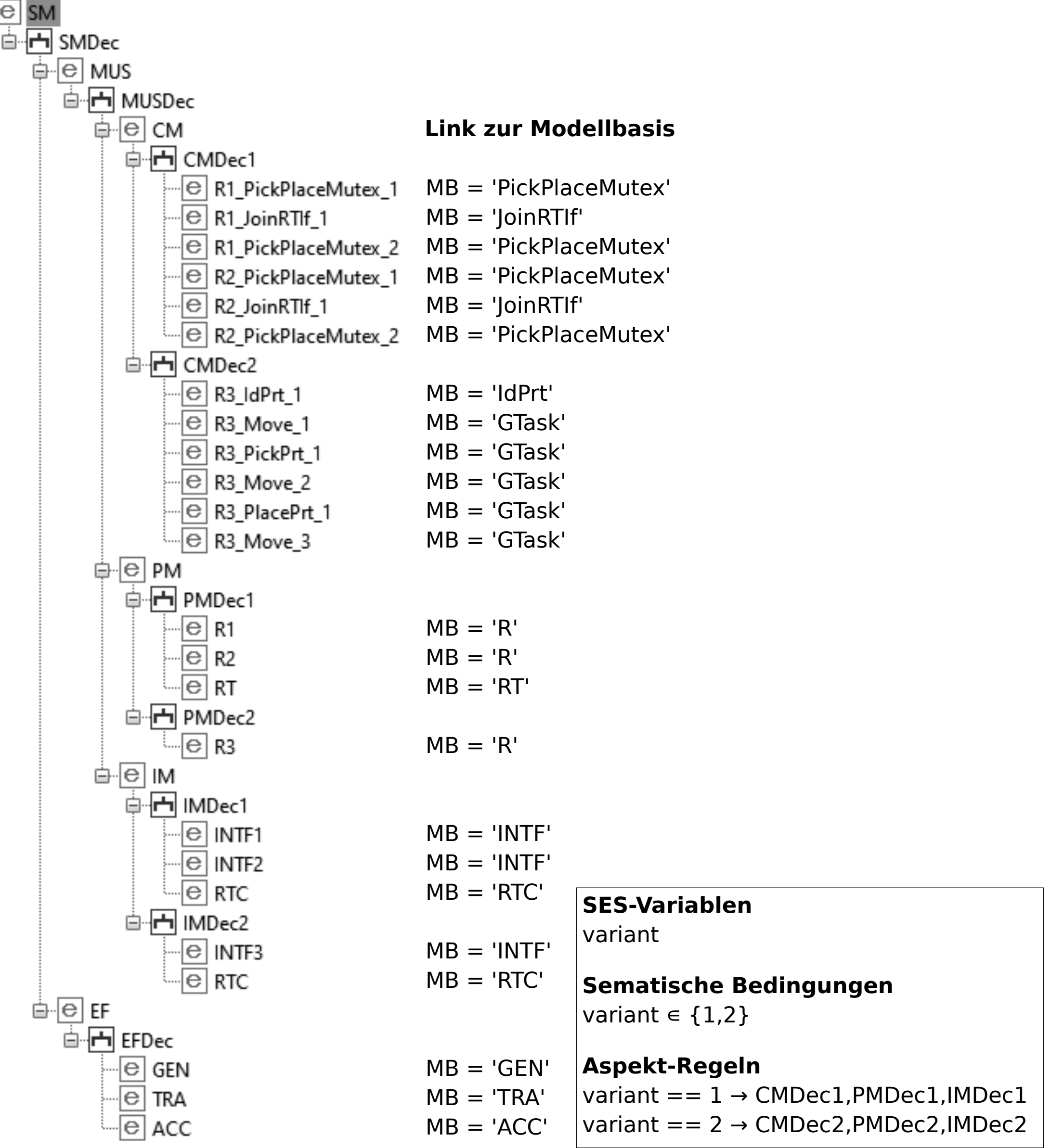


Abbildung 6.16: Lösungsansatz für Interaktionsklasse 6 mit einer SES

Das Simulationsmodell (SM), die spätere Control-Software (CS), ist in zwei Teilmodelle strukturiert. Das Model-Under-Study (MUS) definiert die möglichen Steuerungskonfigurationen. Der Experimental-Frame (EF) definiert einen Untersuchungsrahmen für das MUS sowie Schnittstellen zur Execution-Control (EC). Basierend auf den Ausführungen zum SBC in Abschnitt 2.3.3 ist das MUS in ein Control-Model (CM), Prozessmodell (PM) und

Interfacemodell (IM) unterteilt. Die zwei Steuerungskonfigurationen werden in der SES durch Aspekt-Knoten gleicher Hierachieebene, sogenannte Aspect-Siblings, modelliert. Die Auswahl eines konkreten Aspekt-Knotens erfolgt durch spezifizierte Aspekt-Regeln. Die Definition dieser Regeln basiert auf der SES-Variablen *variant*. Die SES-Variable kann zwei zulässige Werte annehmen. Bei Auswertung der Aspekt-Regeln führt die aktuelle Wertebelung zur Auswahl einer Steuerungskonfiguration. Die Blattknoten der SES spezifizieren mit ihrem MB-Attribut einen Link auf eine Komponente in der Modellbibliothek (MB). Nachfolgend werden die in der SES spezifizierten Konfigurationen diskutiert.

Abbildungen 6.17 und 6.18 zeigen die zwei in der SES spezifizierten Konfigurationen des MUS.

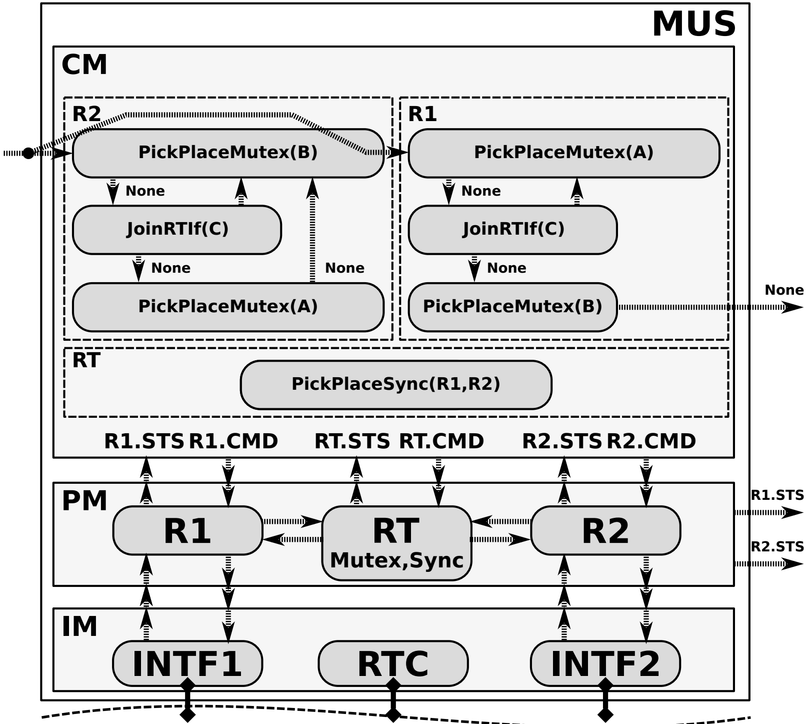


Abbildung 6.17: Steuerungskonfiguration mit zwei Robotern für die Bauteile A, B und C

Abbildung 6.17 spezifiziert eine Robotersteuerung mit den Robotern R1 und R2, welche die Teilearten A,B und C handhaben können. Die Roboter müssen zur Lösung der Aufgabenstellung miteinander interagieren. Eine Handhabung der Teileart D kann durch die Roboter nicht realisiert werden. Zur Handhabung der Teileart D wird die in Abbildung 6.18 dargestellte Konfiguration im Zusammenspiel mit Roboter R3 verwendet.

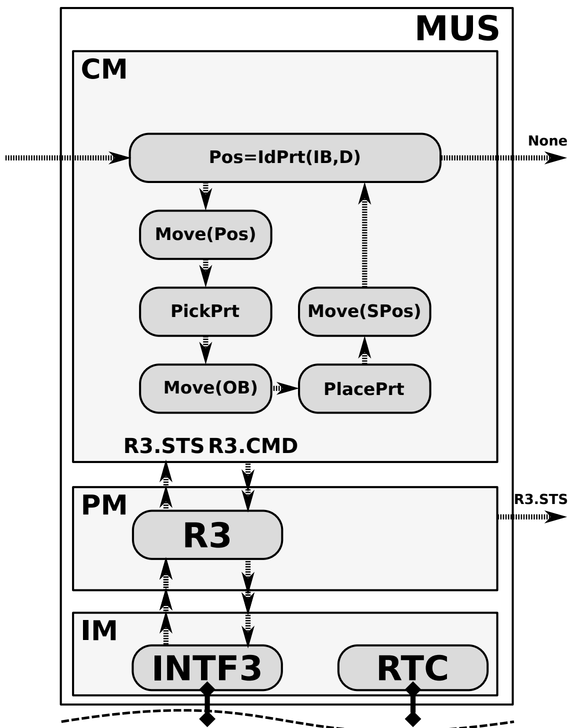


Abbildung 6.18: Steuerungskonfiguration mit Spezialroboter R3 für Bauteil D

Abbildung 6.19 zeigt die Struktur des in der SES spezifizierten EF. Die Struktur des EF ist für beide Steuerungskonfigurationen identisch. Der GEN definiert einen Ausgangsport über den die Abarbeitung der ersten Aufgabe durch die Roboter angestoßen wird. Der TRA berechnet aus den Ausgangsereignissen des MUS Bewertungsgrößen. In diesem Fall wird nur die Anzahl der von den Robotern abgearbeiteten Aufgaben gezählt. Der TRA definiert zwei Ausgangsports. Der Ausgangsport STP wird zur Kommunikation mit der Komponente ACC verwendet. Der ACC stoppt zustandsabhängig die Ausführung der aktuellen Abarbeitung (Simulationslauf oder Steuerung). Weiterhin definiert der TRA einen Ausgangsport RES über den relevante Prozessgrößen an die übergeordnete EU übergeben werden.

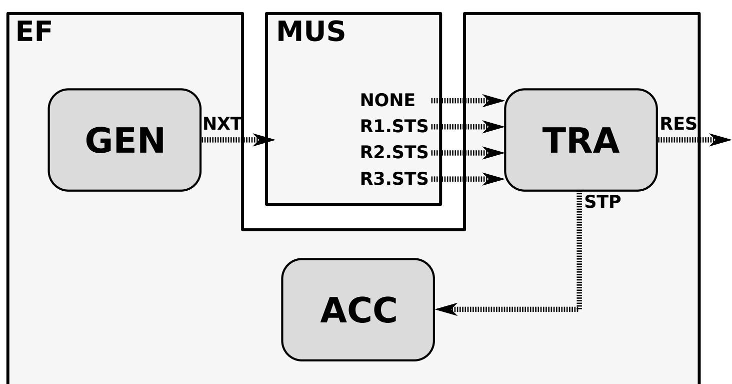


Abbildung 6.19: Struktur des Experimental-Frame

Abbildungen 6.20 zeigt die Umsetzung der Komponente GEN. Diese ist analog der in Abschnitt 6.2.1 eingeführten Aufgabe NoOp umgesetzt. Über die in der SES spezifizierten

Kopplungsrelationen wird die Aufgabenabarbeitung durch Versenden eines Ausgangsereignisses an MUS initiiert. Anschließend wechselt die Komponente GEN vom Zustand Active in den Zustand Passive.

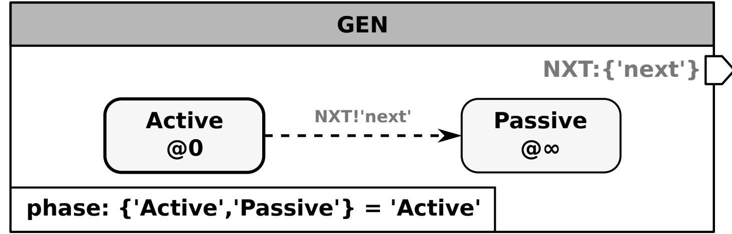


Abbildung 6.20: Erweitertes DEVS-Diagramm der Komponente GEN

Die Umsetzung der Komponente TRA zeigt Abbildung 6.21. Der TRA spezifiziert für jede Roboterkomponente einen Eingangsport *Rx_STS*, um 'done'-Ereignisse zu zählen. Die 'done'-Ereignisse werden jeweils nach erfolgreicher Abarbeitung einer Aufgabe verschickt und stellen einen Performance-Indikator dar. Erhält der TRA über den Eingangsport NONE ein 'next'-Ereignis, so wechselt dieser vom Initialzustand Active in den Zustand Passive. Hierbei werden an den zwei Ausgangsports Ereignisse generiert. Am Ausgangsport RES wird die Anzahl der 'done'-Ereignisse ausgegeben und am Ausgangsport STP wird ein Ereignis 'stp' versendet.

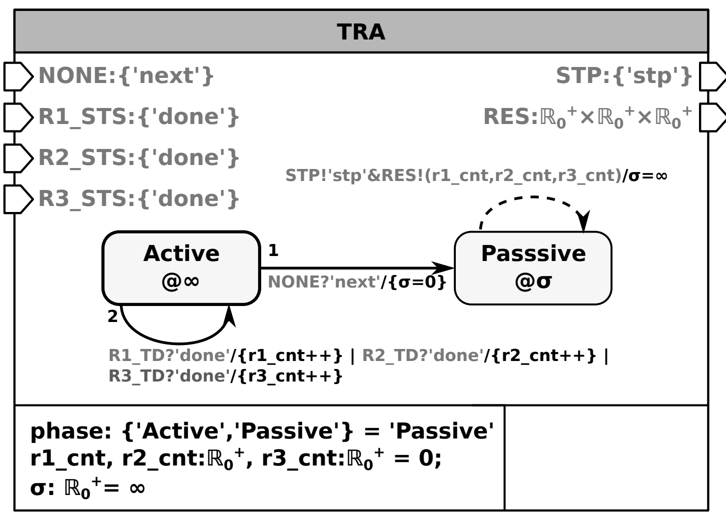


Abbildung 6.21: Erweitertes DEVS-Diagramm der Komponente TRA

Die Spezifikation der Komponente ACC ist in Abbildung 6.22 dargestellt. Der ACC definiert einen Mechanismus zum zustandsbasierten Abbruch der aktuellen Steuerungsausführung. Dieser kann sich simulatorabhängig unterscheiden. Im vorliegenden Beispiel wird die globale Variable SIMUSTOP=TRUE gesetzt. Diese signalisiert dem zugrundeliegenden PDEVS-Simulator, dass die Simulation zu beenden ist.

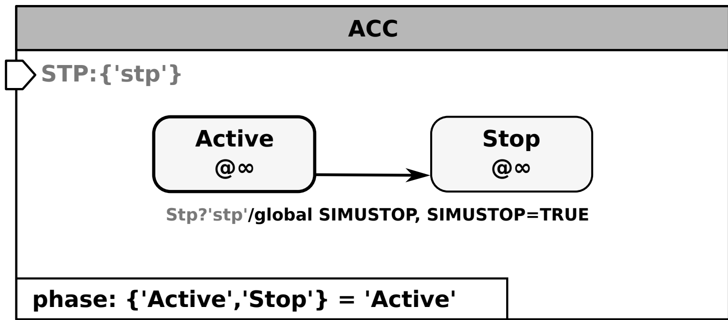


Abbildung 6.22: Erweitertes DEVS-Diagramm der Komponente ACC

In Kapitel 5 wurde die Ausführung SES/MB-basierter Steuerungskonfigurationen erläutert und in Abschnitt 6.1.4 konkretisiert. Demgemäß ist zur Ausführung einer Steuerung eine Execution-Control (EC) zu definieren. Listing 6.1 zeigt den Pseudocode der EC für das betrachtete Beispiel. Die EC definiert mit *variant=1* (Zeile 2) die initiale Steuerungskonfiguration, bestehend aus einem MUS gemäß Abbildung 6.17 und einem EF gemäß Abbildung 6.19. Die Ableitung der aktuellen Konfiguration erfolgt durch Pruning der SES (Zeile 5), woraufhin als Ergebnis eine Pruned-Entity-Structure (PES) zurückgegeben wird. Unter Verwendung der in der SES definierten Links zu Komponenten in der MB, wird mit der build-Methode (Zeile 6) ein SM generiert. Das SM wird an die Execution-Unit (EU), den PDEVS-Simulator, übertragen (Zeile 7) und zur Ausführung gebracht (Zeile 8). Die Ausführung der aktuellen SM-Konfiguration wird bei Erreichen des zustandsbedingten Abbruchkriteriums durch den ACC beendet und die vom TRA gelieferten Ergebnisse werden an die EC zurückgegeben und gespeichert (Zeile 9). Die EC kann durch eine Wertzuweisung an die SES-Variable *variant* (Zeile 11-14) die Generierung und Ausführung einer neuen Steuerungskonfiguration veranlassen oder sich selbst beenden und Resultate zurückgeben (Zeile 15-17).

Listing 6.1: Pseudocode der Execution-Control (EC)

```
1  Results = [ ];
2  variant = 1;
3  cnt = 0;
4  while TRUE:
5    PES = pruning(SES,variant)
6    SM = build(PES);
7    EU.transmit(SM);
8    RES = EU.execute();
9    Results.Add(RES);
10   cnt++;
11   if variant == 2 :
12     variant = 1;
13   else
14     variant = 2;
15   if  cnt == 10 :
16     breakLoop;
17 Results.display();
```

Die Steuerungsumsetzung unter Verwendung des Erweiterten SES/MB-Frameworks führt zu einer Reduzierung der Komplexität der SBC-basierten Modelle. Die Implementierung,

der Test und die Wartung der Modelle vereinfacht sich. Andererseits erfordert der Ansatz die Implementierung neuer Komponenten und einen höheren Einarbeitungsaufwand.

6.3 Zusammenfassung

Im ersten Abschnitt wurde eine SBC-Plattform für kooperierende Robotersysteme auf Basis von MATLAB vorgestellt. Die Kommunikation mit realen und virtuellen Robotern basiert auf der *Robotic Control and Visualization-(RCV)* Toolbox, einer roboterorientierten Middleware. Diese stellt das Interface zwischen der Hardware und dem IM des SBC-Ansatzes dar. Die einzelnen Komponenten des SBC-Konzepts – auf der IM-, PM- und CM-Ebene – wurden mit der *DEVS Toolbox for MATLAB* implementiert. Die DEVS Toolbox ermöglichte eine effiziente Umsetzung des PDEVS-RCP-V2-Formalismus in der interaktiven MATLAB-Umgebung. Mit der Softwarearchitektur konnte der praktische Nachweis der Implementierung aufgabenorientierter Steuerungen (TOC) nach dem SBC-Ansatz erbracht werden. Das im Kapitel 5 eingeführte *Erweiterte SES/MB-Framework* zur Implementierung flexibler Steuerungen wurde als Softwareschicht oberhalb des SBC-Ansatzes eingeführt. Der praktische Nachweis des Konzepts wurde mit einer softwaretechnischen Realisierung unter Nutzung der *SES-Toolbox for MATLAB* erbracht.

Im zweiten Abschnitt dieses Kapitels wurden ausgewählte Lösungsansätze nach Abschnitt 4.3, unter Verwendung der im Abschnitt 3.2.3 eingeführten DEVS-Diagramme, konkretisiert. Die softwaretechnische Umsetzung erfolgte auf Basis des PDEVS-RCP-V2-Formalismus mit der DEVS Toolbox for MATLAB. Im Rahmen der Umsetzung wurden generische Modellkomponenten abgeleitet, welche durchgehend wiederverwendet wurden. Weiterhin wurde gezeigt, dass Modelle vorangegangener Lösungsansätze schrittweise weiterentwickelt werden können. Für das Beispiel zur Interaktionsklasse 6, welches eine flexible Steuerung beinhaltet, wurden zwei Lösungsvarianten implementiert. Die erste Variante entspricht einer 150%-Modellierung gemäß der Begriffsdefinition in Abschnitt 5.2. Beim zweiten Lösungsansatz erfolgte die Implementierung als flexible Steuerung auf Basis der Integration von SBC-Ansatz und SES/MB-Framework. Die Steuerungskonfiguration ist zur Laufzeit jeweils auf die aktuell zu lösende Steuerungsaufgabe zugeschnitten. Dadurch ist die Komplexität der Steuerungsstruktur zur Laufzeit auf das aktuell unbedingt notwendige Maß reduziert. Ist die aktuelle Steuerungsaufgabe abgeschlossen, wird für die nächste Aufgabe eine neue maßgeschneiderte Steuerungskonfiguration generiert und zur Ausführung gebracht. Die Entwicklung des notwendigen Software-Frameworks ist aufwendig, aber es ist ein einmaliger Aufwand und das Framework ist vielseitig einsetzbar. Aufgrund der Modularität des Ansatzes wird die Entwicklung von flexiblen Steuerungen, gemäß dem Beispiel zur Interaktionsklasse 6, wesentlich erleichtert. Im Gegensatz zur 150%-Modellierung wird nicht eine komplexe monolithische Steuerungskonfiguration entwickelt. Die Steuerungsentwicklung erfolgt schrittweise auf Basis einzelner Module, die unabhängig voneinander getestet werden können. Weiterhin kann eine flexible Steuerung schrittweise und modular in Betrieb genommen werden. Das heißt, dass sich die einzelnen Steuerungskonfigurationen separat testen lassen, wodurch die Wartbarkeit der gesamten Steuerung erleichtert wird. Die Entscheidung, wann der 150%-Ansatz oder der zweite Ansatz auf Basis des SES/MB-Frameworks mit maßgeschneiderten Steuerungskonfigurationen zum Einsatz kommen soll, muss je nach Anwendungsfall individuell bewertet werden.

7 Zusammenfassung und Ausblick auf weiterführende Arbeiten

Die vorliegende Arbeit ist unter der Zielstellung entstanden, einen Beitrag zur durchgängigen, modellbasierten und herstellerunabhängigen Steuerungsentwicklung für gelenkarmbasierte Multi-Robotersysteme (MRS) zu leisten. Als Ausgangspunkt dienten langjährige Vorarbeiten der Forschungsgruppe CEA. Als modellbasierter Ansatz wurden der DEVS-Formalismus und als Vorgehensmodell sowie Software-Framework der SBC-Ansatz ausgewählt. Für die modulare Steuerungsbeschreibung wurde auf den aufgabenorientierten Steuerungsentwurf aufgebaut. Vollständiges Neuland ist die abstrakte Beschreibung von Interaktionen zwischen Robotern in Form von modularen, aufgabenorientierten Aktionen. Die abstrakte Beschreibung von Interaktionen ist wesentlich für die aufgabenorientierte Steuerungsentwicklung für MRS. Nicht betrachtet wurden Interaktionen zwischen Robotern und Menschen sowie der Umwelt.

Die Arbeit startet mit einer Analyse unterschiedlicher Paradigmen zur Steuerungsentwicklung. Weiterhin erfolgt eine kurze Betrachtung möglicher Middleware-Konzepte, welche eine Integration unterschiedlicher Robotersysteme ermöglichen. Anschließend wurden verschiedene Methoden zur Online- und Offline-Programmierung von Robotersystemen sowie Vorgehensmodelle und Entwicklungs-Frameworks diskutiert. Auf Grundlage der Analyse wurde die MATLAB-basierte RCV-Toolbox als Middleware und eine aufgabenorientierte Steuerungsentwicklung nach dem SBC-Vorgehensmodell als Basis für die weiteren Untersuchungen ausgewählt. Demgemäß wurde der Stand der Technik hinsichtlich der Entwicklung von aufgabenorientierten Steuerungen nach dem SBC-Ansatz untersucht. Dabei zeigte sich, dass die bisherigen Arbeiten sich ausschließlich auf Single-Robotersysteme beziehen und es bisher keine Arbeiten zur Steuerungsentwicklung für MRS in diesem Kontext gibt. Auf Basis der Literatur erfolgte eine Charakterisierung von MRS. Der Schwerpunkt wurde auf den Aspekt der Interaktion gelegt und es wurden sechs Interaktionsklassen als Basis für die weitere Betrachtung spezifiziert.

Eine Steuerungsentwicklung für MRS erfordert die Umsetzung vielfältiger Anforderungen und ausführliche Tests, woraus die Zweckmäßigkeit des modellbasierten Entwicklungsansatzes folgt. Der Entwicklungsprozess erfolgt in Phasen, welche separate Teilziele verfolgen. Aus Effizienzgründen besteht der Wunsch, ein entwickeltes Modell phasenübergreifend zu nutzen und weiter zu entwickeln. Um dieser Anforderung gerecht zu werden, wurde der DEVS-Formalismus hinsichtlich des Einsatzes in der Steuerungsentwicklung analysiert. Als Modellierungs- und Simulationsformalismus kann er in der Entwurfsphase originär angewendet werden. Beim Übergang in die Automatisierungsphase wird eine Echtzeit- und Prozessanbindung erforderlich. In diesem Zusammenhang wurden fünf aus der Literatur bekannte, erweiterte DEVS-Formalismen untersucht. Drei der untersuchten Formalismen

erfordern einen Austausch des Simulators beim Übergang von der Entwurfs- zur Automatisierungsphase und waren für die Ziele dieser Arbeit ungeeignet. Die zwei verbliebenen Formalismen erfordern keinen Tausch des Simulators. Die Integration externer Hardwarekomponenten wird auf Modellebene umgesetzt. Dabei wird der Zustand externer Hardwarekomponenten durch die Modellkomponenten mittels Pollen bestimmt. Dies führt zum Einplanen von einer Vielzahl interner Ereignisse und somit zu Overhead. Weiterhin ist die vorgegebene Umsetzung auf Modellebene teilweise sehr restriktiv.

Aufbauend auf der Analyse wurde entsprechend dem PDEVS-RCP Formalismus ein modifizierter Ansatz (PDEVS-RCP2-V2) entwickelt, welcher insbesondere darauf abzielt, den analysierten Overhead und die Modellanpassungen zu reduzieren. Es konnte gezeigt werden, dass sich der neue Formalismus zur durchgängigen Entwicklung aufgabenorientierter Steuerungen auf Basis des SBC-Ansatzes eignet und sich der Aufwand hinsichtlich der Modellanpassungen beim Übergang in die Automatisierungs- und Betriebsphase erheblich reduziert. Anhand eines häufig verwendeten Fallbeispiels wurden die Vorteile des neuen Formalismus gegenüber der Vorgängerversion aufgezeigt. Weiterhin wurde gezeigt, dass sich Interaktionen in MRS modular und aufgabenorientiert beschreiben lassen. Die Untersuchung erfolgte auf Grundlage der zuvor definierten Interaktionsklassen. Es wurde für jede Klasse schrittweise eine aufgabenorientierte Spezifikation entwickelt. Dabei zeigte sich, dass bekannte Prinzipien zur Interprozesskommunikation bei Betriebssystemen sehr gut auf MRS übertragbar sind. Damit können die ersten drei in der Zielstellung formulierten Hypothesen konstatiert werden.

Hypothese 1: *Interaktionen zwischen Gelenkarmrobotern in einem MRS können mittels wiederverwendbarer und komponierbarer Aufgaben beschrieben werden.*

Hypothese 2: *Mit dem DEVS-Formalismus kann für MRS eine durchgängige modellbasierte Steuerungsentwicklung erfolgen.*

Hypothese 3: *Der SBC-Ansatz als Vorgehensmodell sowie Software-Framework unterstützt eine systematische Steuerungsentwicklung für MRS und in Verbindung mit einer entsprechenden Middleware eine herstellerunabhängige Entwicklung.*

Weiterhin zeigte sich bei der Entwicklung der Lösungsvarianten, dass der Schwierigkeitsgrad der aufgabenorientierten Spezifikation und der Aufgabentransformation durch Interaktionen schnell anwächst. Insbesondere der Lösungsansatz zur sechsten Interaktionsklasse zeigte, dass die Steuerungsspezifikation und die Aufgabentransformation aufgrund der Flexibilität der Steuerung eine Herausforderung für Entwickler darstellt. Aus diesem Grund wurde nach einem weiteren Lösungsansatz gesucht. Dabei wurde auf den Vorschlag von Maletzki, das System-Entity-Structure/Model-Base (SES/MB) Framework mit Bezug auf aufgabenorientierte Steuerungen und dem SBC-Ansatz zu untersuchen, zurückgegriffen. Die Grundlagen des SES/MB-Ansatzes sowie spezifische Erweiterungen wurden analysiert. Darauf aufbauend wurde ein angepasstes SES/MB-basiertes Framework für flexible Steuerungen abgeleitet. Mit dem entwickelten Framework kann eine Modifikation der Steuerungskonfiguration zur Laufzeit einer Steuerung erfolgen. Die zugrundeliegende Steuerungsstruktur basiert auf dem zuvor eingeführten SBC-Ansatz. Des Weiteren war es wichtig, den Kontext einer Steuerungskonfiguration, dass heißt Zielstellungen, Randbedingungen, Abbruchbedingungen etc., zu modellieren. Hierzu wurde auf das Konzept des

Experimental-Frame zurückgegriffen und es wurde dessen Anwendung im Zusammenhang mit dieser Arbeit erläutert.

Abschließend wurden die eingeführten Konzepte und Methoden soweit konkretisiert, dass deren prototypische softwaretechnische Umsetzung aufgezeigt werden kann. Hierfür wurde MATLAB als eine SBC-Plattform für kooperierende Robotersysteme eingeführt. Die Kommunikation mit realen und virtuellen Robotern basiert auf der *Robotic Control and Visualization (RCV)* Toolbox, einer herstellerunabhängigen Middleware. Die RCV-Toolbox stellt das Interface zwischen der Hardware und der echtzeitsynchronisierten, simulationsbasierten Steuerung dar. Die einzelnen Ebenen des SBC-Konzepts wurden softwaretechnisch mit der *DEVS Toolbox for MATLAB* realisiert. Die Toolbox ermöglichte eine effiziente Umsetzung des in der Arbeit entwickelten PDEVS-RCP-V2-Formalismus. Damit konnte die gesamte Softwarearchitektur zur Entwicklung aufgabenorientierter Steuerungen nach dem SBC-Ansatz programmtechnisch umgesetzt werden. Das im Kontext flexibler Steuerungen eingeführte *Erweiterte SES/MB-Framework* wurde als Softwareschicht oberhalb des SBC-Ansatzes eingeführt. Die softwaretechnische Realisierung erfolgte durch Adaption der *SES-Toolbox for MATLAB*. Die gegenüberstellende Umsetzung eines Fallbeispiels bestätigt die vierte in der Zielsetzung formulierte Hypothese.

Hypothese 4: *Auf Basis des SES/MB-Frameworks können strukturiert flexible (adaptive) Steuerungen für MRS realisiert werden.*

Den Kern der Arbeit bildete die Untersuchung der TOC-basierten Lösungsansätze für MRS auf Basis der sechs Interaktionsklassen. Die Umsetzung erfolgte auf Grundlage des modifizierten DEVS-Formalismus. Zur Darstellung der Modelle wurde eine *Erweiterte DEVS-Diagramm-Notation* neu eingeführt. Im Rahmen der Umsetzung wurden generische Modellkomponenten abgeleitet, welche durchgehend wiederverwendet wurden. Weiterhin wurde gezeigt, dass Modelle vorangegangener Lösungsansätze schrittweise weiterentwickelt werden können. Für das Beispiel zur sechsten Interaktionsklasse, welches eine flexible Steuerung darstellt, wurden zwei Lösungsvarianten entwickelt. Die erste Variante entspricht einer 150%-Modellierung, gemäß der Definition im Softwareengineering (SE). Die Steuerungskonfiguration umfasst eine hohe Anzahl an Aufgabenmodulen, wobei zeitgleich nur wenige Aufgabenmodule aktiv sind. Zur Laufzeit wird zwischen den Aufgabenmodulen umgeschaltet. Die Implementierung der Steuerung zeigte, dass die Wartbarkeit, der Test und die Inbetriebnahme der Steuerung in Form einer monolithischen Steuerungskonfiguration fehleranfällig und zeitaufwendig ist. Ein Ziel dieser Arbeit ist die systematische Steuerungsentwicklung für MRS zu ermöglichen, weshalb der Wartbarkeit und dem Test der Steuerung eine besondere Bedeutung zukommt. Durch eine monolithische Steuerungskonfiguration wird die Umsetzung dieses Ziel erschwert, sodass ein zweiter Lösungsansatz untersucht wurde. Beim zweiten Lösungsansatz erfolgte die Realisierung als flexible Steuerung auf Basis der Integration von SBC-Ansatz und SES/MB-Framework. Die Steuerungskonfiguration ist zur Laufzeit auf die aktuell zu lösende Steuerungsaufgabe zugeschnitten. Dadurch wird die Komplexität der Steuerungsstruktur zur Laufzeit auf eine aktive Steuerungskonfiguration reduziert. Ist die aktuelle Steuerungsaufgabe abgeschlossen, wird eine neue Steuerungskonfiguration zur Lösung der nächsten Aufgabe generiert und zur Ausführung gebracht. Die Wartbarkeit, der Test und die Inbetriebnahme der Steuerung wird durch die klare Strukturierung und die Reduktion der zur Laufzeit aktiven Konfiguration erleichtert.

In der Arbeit wurde der aufgabenorientierte Steuerungsentwurf als geeignetes Beschrei-

bungsmittel für die Spezifikation von Interaktionen zwischen Industrierobotern eingeführt. Im Kontext der Robotik treten Interaktionen weiterhin zwischen Menschen – Robotern auf. In diesem Zusammenhang wäre die Übertragung der entwickelten Konzepte auf diesen Anwendungsbereich zu überprüfen. Weiterhin wäre es interessant, ob sich spezifische Interaktionen zwischen Mensch und Roboter identifizieren und durch eine aufgabenorientierte Steuerung auf Basis des SBC-Ansatzes abbilden lassen. Ebenso wurde die Frage, ob die Ausführung einer bestimmten Aufgaben -und Interatkionsfolge eine gegebene Steuerungsaufgabe in optimaler Weise löst, nicht untersucht. Die Steuerungsstrategie könnte zur Laufzeit unter Berücksichtigung gegebener Randbedingungen und Umwelteinflüsse erlernt werden und sich situativ ändert. Der zweite Lösungsansatz zu Interaktionsklasse sechs zeigt mit dem *Erweiterten SES/MB-Ansatz* ein Werkzeug mit dessen Hilfe sich diese Fragestellung in weiterführenden Arbeiten untersuchen lässt.

Literaturverzeichnis

[1] Abel, D. ; Bollig, A.: *Rapid Control Prototyping, Methoden und Anwendung.* Springer, 2006

[2] Alpar, Paul ; Grob, Heinz L. ; Weimann, Peter ; Winter, Robert: *Anwendungsorientierte Wirtschaftsinformatik: Strategische Planung, Entwicklung und Nutzung von Informations- und Kommunikationssystemen.* 5., überarb. und aktualisierte Aufl. Wiesbaden : Vieweg, 2008 http://deposit.ddb.de/cgi-bin/dokserv?id=2606477&prov=M&dok_var=1&dok_ext=htm. – ISBN 978–3–8348–0438–9

[3] Andreas Burth, Marc G. ; HaushaltsSteuerung.de (Hrsg.): *Lexikon zur öffentlichen Haushalts- und Finanzwirtschaft: Aktivitätsniveau - passiv, reaktiv, aktiv, proaktiv.* https://www.haushaltssteuerung.de/lexikon-aktivitaetsniveau-passiv-reaktiv-aktiv-proaktiv.html

[4] Auer, Michael E. (Hrsg.) ; Ram B., Kalyan (Hrsg.): *Cyber-physical Systems and Digital Twins.* Cham : Springer International Publishing, 2020 (Lecture Notes in Networks and Systems). http://dx.doi.org/10.1007/978-3-030-23162-0. http://dx.doi.org/10.1007/978-3-030-23162-0. – ISBN 978–3–030–23161–3

[5] Ayed, M.B. ; Zouari, L. ; Abid, M.: Software In the Loop Simulation for Robot Manipulators. In: *Engineering, Technology & Applied Science Research* 7 (2017), Oct., Nr. 5, 2017–2021. http://dx.doi.org/10.48084/etasr.1285. – DOI 10.48084/etasr.1285

[6] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb.* Springer Spektrum, 2011. – ISBN 978–3–8274–1706–0

[7] Bartelt, Matthias ; Stumm, Sven ; Kuhlenkötter, Bernd: Tool Oriented Robot Cooperation. In: *Procedia CIRP* 23 (2014), S. 188–193. http://dx.doi.org/10.1016/j.procir.2014.10.103. – DOI 10.1016/j.procir.2014.10.103. – ISSN 22128271

[8] Behnke, Sven: Kooperierende Mobile Roboter. In: *at - Automatisierungstechnik* 61 (2013), Nr. 4, S. 233–244. http://dx.doi.org/10.1524/auto.2013.0016. – DOI 10.1524/auto.2013.0016. – ISSN 0178–2312

[9] Bergero, Federico ; Kofman, Ernesto: PowerDEVS: a tool for hybrid system modeling and real-time simulation. In: *SIMULATION* 87 (2011), Nr. 1-2, S. 113–132. http://dx.doi.org/10.1177/0037549710368029. – DOI 10.1177/0037549710368029. – ISSN 0037–5497

[10] Bielawny, Dirk ; Bruns, Torsten ; Loh, Chia C. ; Traechtler, Ansgar: Multi-Robot Approach for Automation of an Industrial Profile Lamination Process. In: *Procedia Engineering* 41 (2012), 981-987. http://dx.doi.org/https://doi.org/10.1016/j.proeng.2012.07.272. – DOI https://doi.org/10.1016/j.proeng.2012.07.272. – ISSN 1877–7058. – International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012)

[11] Bilberg, Arne ; Malik, Ali A.: Digital twin driven human–robot collaborative assembly. In: *CIRP Annals* 68 (2019), Nr. 1, S. 499–502. http://dx.doi.org/10.1016/j.cirp.2019.04.011. – DOI 10.1016/j.cirp.2019.04.011. – ISSN 00078506

[12] Birger Freymann: *Entwicklung einer Modellbibliothek für die Interaktion von Robotern in der MATLAB/DEVS Umgebung*. Wismar, Wismar, Masterthesis, 10.06.2014

[13] Bonaventura, Matías ; Wainer, Gabriel A. ; Castro, Rodrigo: Graphical modeling and simulation of discrete-event systems with CD++Builder. In: *SIMULATION* 89 (2013), Nr. 1, S. 4–27. http://dx.doi.org/10.1177/0037549711436267. – DOI 10.1177/0037549711436267. – ISSN 0037–5497

[14] Capocchi, L. ; Santucci, J. F. ; Poggi, B. ; Nicolai, C.: DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems. In: *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE, 27.06.2011 - 29.06.2011. – ISBN 978–1–4577–0134–4, S. 170–175

[15] Chandra Karmokar, Bikash ; Durak, Umut ; Jafer, Shafagh ; Chhaya, Bharvi N. ; Hartmann, Sven: Tools for Scenario Development Using System Entity Structures. In: *AIAA Scitech 2019 Forum*. Reston, Virginia : American Institute of Aeronautics and Astronautics, 01072019. – ISBN 978–1–62410–578–4

[16] Chinello, Francesco ; Scheggi, Stefano ; Morbidi, Fabio ; Prattichizzo, Domenico: KUKA Control Toolbox. In: *IEEE Robotics & Automation Magazine* 18 (2011), Nr. 4, S. 69–79. http://dx.doi.org/10.1109/MRA.2011.942120. – DOI 10.1109/MRA.2011.942120. – ISSN 1070–9932

[17] Cho, Seong M. ; Kim, Tag G.: Real Time Simulation Framework for RT-DEVS Models. In: *Trans. Soc. Comput. Simul. Int.* 18 (2001), Nr. 4, 203–215. http://dl.acm.org/citation.cfm?id=637757.637760. – ISSN 0740–6797

[18] Cho, Young K. ; Hu, Xiaolin ; Zeigler, Bernard P.: The RTDEVS/CORBA Environment for Simulation-Based Design of Distributed Real-Time Systems. In: *SIMULATION* 79 (2003), Nr. 4, 197–210. http://dx.doi.org/10.1177/0037549703038880. – DOI 10.1177/0037549703038880. – ISSN 0037–5497

[19] Chow, A. C.-H.: Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism and Its Distributed Simulator. In: *Trans. Soc. Comput. Simul. Int.* 13 (1996), Nr. 2, 55–67. http://dl.acm.org/citation.cfm?id=240054.240057. – ISSN 0740–6797

[20] Christern, Michael ; Schmidt, Artur ; Schwatinski, Tobias ; Pawletta, Thorsten: *KUKA-KAWASAKI-Robotic Toolbox for Matlab*. http://www.cea-wismar.de/tbx/KK_Robotic_Tbx/KK_Robotic_Tbx.html. Version: 2011

[21] Christina Deatcu ; Birger Freymann ; Artur Schmidt ; Thorsten Pawletta: *MATLAB/Simulink Based Rapid Control Prototyping for Multivendor Robot Applications*

[22] Corrales, J. A. ; Gómez, G. J. G. ; Torres, F. ; Perdereau, V.: Cooperative Tasks between Humans and Robots in Industrial Environments. In: *International Journal of Advanced Robotic Systems* 9 (2012), Nr. 3, S. 94. http://dx.doi.org/10.5772/50988. – DOI 10.5772/50988. – ISSN 1729–8814

[23] Coulouris, George F. ; Dollimore, Jean ; Kindberg, Tim: *Verteilte Systeme: Konzepte und Design*. 3., überarb. Aufl. München : Pearson Studium, 2002 (Informatik : Verteilte Systeme). – ISBN 3–8273–7022–1

[24] Couretas, J. M. ; Rozenblit, J. W.: Generation, control, and simulation of task level actions based on discrete event models. In: *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*, IEEE Comput. Soc. Press, 7-9 Dec. 1994. – ISBN 0–8186–6440–1, S. 214–220

[25] Czarnecki, C.A.: Multi-Robot Systems: A Perspective. In: *IFAC Proceedings Volumes* 27 (1994), Nr. 4, 201-205. http://dx.doi.org/https://doi.org/10.1016/S1474-6670(17)46023-X. – DOI https://doi.org/10.1016/S1474–6670(17)46023–X. – ISSN 1474–6670. – IFAC Workshop on Intelligent Manufacturing Systems 1994 (IMS'94), Vienna, Austria, 13-15 June

[26] Daum, Thorsten ; Sargent, Robert G.: Experimental frames in a modern modeling and simulation system. In: *IIE Transactions* 33 (2001), Nr. 3, S. 181–192. http://dx.doi.org/10.1080/07408170108936821. – DOI 10.1080/07408170108936821. – ISSN 0740–817X

[27] Deatcu, C. (Hrsg.) ; Freymann, B. (Hrsg.) ; Schmidt, A. (Hrsg.) ; Pawletta, T. (Hrsg.): *MATLAB/Simulink Based Rapid Control Prototyping for Multivendor Robot Applications*. Bd. *25*. Simulation Notes Europe, 2015

[28] Deatcu, C. ; Schwatinski, T. ; Pawletta, T.: *DEVS Toolbox for MATLAB*. http://www.mb.hs-wismar.de/cea/DEVS_Tbx/MatlabDEVS_Tbx.html. Version: 2013

[29] Deatcu, Christina ; Freymann, Birger ; Pawletta, Thorsten: PDEVS-based Hybrid System Simulation Toolbox for MATLAB. In: *Proceedings of the Symposium on Theory of Modeling & Simulation*. San Diego, CA, USA : Society for Computer Simulation International, 2017 (TMS/DEVS '17), 2:1–2:12

[30] Derler, Patricia ; Lee, Edward A. ; Tripakis, Stavros ; Törngren, Martin: Cyber-physical System Design Contracts. In: *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*. New York, NY, USA : ACM, 2013 (ICCPS '13). – ISBN 978–1–4503–1996–6, 109–118

[31] Deutsches Institut für Normung (DIN) e.V.: *Programmiersprache Industrial Robotic Language (IRL)*. Berlin, 09.01.1996

[32] Deutsches Institut für Normung (DIN) e.V.: *Simulation von Logistik-, Materialfluss- und Produktionssystemen*. Düsseldorf, 2014

[33] Durak, Umut ; Pruter, Insa ; Gerlach, Torsten ; Jafer, Shafagh ; Pawletta, Thorsten ; Hartmann, Sven: Using System Entity Structures to Model the Elements of a Scenario in a Research Flight Simulator. In: *AIAA Modeling and Simulation Technologies Conference*. Reston, Virginia : American Institute of Aeronautics and Astronautics, 01092017. – ISBN 978–1–62410–451–0, S. 1278

[34] EASY-ROB: *EASY-ROB 3D Robot Simulation Tool*. http://www.easy-rob.com/de/easy-rob/

[35] Fishwick, P. A. (Hrsg.) ; Modjeski, R. B. (Hrsg.) ; Ören, T. I. (Hrsg.): *Dynamic Templates and Semantic Rules for Simulation Advisors and Certifiers*. Bd. *4*. Springer-Verlag, 1991

[36] Folkerts, Hendrik ; Pawletta, Thorsten ; Deatcu, Christina ; Durak, Umut: Variability Modeling for Engineering Applications. In: *SNE Simulation Notes Europe* 27 (2017), Nr. 4, S. 167–176. http://dx.doi.org/10.11128/sne.27.tn.10391. – DOI 10.11128/sne.27.tn.10391. – ISSN 2305–9974

[37] Fraser, Gordon ; Steinbauer, Gerald ; Wotawa, Franz ; (tug, Technische Universität G.: Application of Qualitative Reasoning to Robotic Soccer. In: *In 18th International Workshop on Qualitative Reasoning*, 2004

[38] Freymann, Birger ; Pawletta, Thorsten ; Pawletta, Sven: Multi-Robotersteuerungen mit variablen Interaktionsprinzipien auf Basis des Simulation Based Control Frameworks und dem Dicrete Event System Specification Formalismus. In: *Proc. of ASIM-Treffen STS/GMMS* (2015), Nr. ARGESIM Report AR 50, 67–77. https://www.researchgate.net/publication/279530602_Multi-Robotersteuerungen_mit_variablen_Interaktionsprinzipien_auf_Basis_des_Simulation_Based_Control_Frameworks_und_dem_Discrete_Event_System_Specification_Formalismus

[39] Freymann, Birger ; Pawletta, Thorsten ; Schwatinski, Tobias ; Pawletta, Sven: Modellbibliothek für die Interaktion von Robotern in der MATLAB/DEVS-Umgebung auf Basis des SBC-Frameworks. In: *ASIM-Workshop STS/GMMS* (2014), 199–208. https://publikationen.reutlingen-university.de/files/338/338.pdf

[40] Goldstein, Rhys ; Breslav, Simon ; Khan, Azam: Practical aspects of the DesignDEVS simulation environment. In: *SIMULATION* 94 (2018), Nr. 4, S. 301–326. http://dx.doi.org/10.1177/0037549717718258. – DOI 10.1177/0037549717718258. – ISSN 0037–5497

[41] Grönninger, Hans ; Krahn, Holger ; Pinkernell, Claas ; Rumpe, Bernhar: Modeling Variants of Automotive Systems using Views. (2014). http://arxiv.org/pdf/1409.6629v1

[42] Groover, Mikell P.: *Automation, Production Systems, and Computer-Integrated Manufacturing*. 2nd edition. Prentice Hall, 2003. – ISBN 0–13–089546–6

[43] Gunar Maletzki, Michael Christern, Artur Schmidt, Thorsten Pawletta, Peter Dünow: *KUKA-KRL-Toolbox for Matlab and Scilab.* `http://www.cea-wismar.de/tbx/Kuka_KRL_Tbx/Kuka_KRL_Tbx.html`. Version: 2011

[44] Haber, Arne ; Kolassa, Carsten ; Manhart, Peter ; Nazari, Pedram Mir S. ; Rumpe, Bernhard ; Schaefer, Ina: First-class variability modeling in Matlab/Simulink. In: Gnesi, Stefania (Hrsg.) ; Collet, Philippe (Hrsg.) ; Schmid, Klaus (Hrsg.): *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems - VaMoS '13.* New York, New York, USA : ACM Press, 2013. – ISBN 9781450315418, S. 1

[45] Hagendorf, O. ; Pawletta, T. ; Deatcu, C.: Extended Dynamic Structure DEVS. In: *Proc. of 21th European Modelling & Simulation Symposium, Puerto de la Cruz, Spain* (2009), Nr. Volume 1, S. 36–45

[46] Hammerschall, Ulrike: *Verteilte Systeme und Anwendungen: Architekturkonzepte, Standards und Middleware-Technologien.* München u.a. : Addison-Wesley Verlag and Pearson Studium, 2005 (Informatik : Verteilte Systeme). – ISBN 3–8273–7096–5

[47] Hirsch-Kreinsen, Hartmut ; Karačić, Anemari: *Autonome Systeme und Arbeit: Perspektiven, Herausforderungen und Grenzen der Künstlichen Intelligenz in der Arbeitswelt.* transcript Verlag, 2019. – ISBN 9783837643954

[48] Hong, Joon S. ; Song, Hae-Sang ; Kim, Tag G. ; Park, Kyu H.: A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development. In: *Discrete Event Dynamic Systems* 7 (1997), Nr. 4, 355–375. `http://dx.doi.org/10.1023/A:1008262409521`. – DOI 10.1023/A:1008262409521. – ISSN 09246703

[49] Hrúz, Branislav ; Zhou, MengChu: *Modeling and Control of Discrete-event Dynamic Systems: With Petri Nets and Other Tools.* London : Springer-Verlag, 2007 (Advanced Textbooks in Control and Signal Processing). `http://dx.doi.org/10.1007/978-1-84628-877-7`. `http://dx.doi.org/10.1007/978-1-84628-877-7`. – ISBN 9781846288722

[50] Kang, Bong G. ; Seo, Kyung-Min ; Kim, Tag G.: Model-Based Design of Defense Cyber-Physical Systems to Analyze Mission Effectiveness and Network Performance. In: *IEEE Access* 7 (2019), S. 42063–42080. `http://dx.doi.org/10.1109/ACCESS.2019.2907566`. – DOI 10.1109/ACCESS.2019.2907566

[51] Kecher, Christoph: *UML 2.0: Das umfassende Handbuch ; [aktuell zum UML-Standard 2.0 ; alle Diagramme und Notationselemente ; Praxisbeispiele in C# und Java 5 ; inkl. CD mit UML-Tools ; inlk. DIN A2-Poster mit Struktur- und Verhaltensdiagrammen.* 1. Aufl., 1. Nachdr. Bonn : Galileo Press, 2005 (Galileo computing). – ISBN 3–89842–573–8

[52] Kim, Sungung ; Sarjoughian, Hessam S. ; Elamvazhuthi, Vignesh: DEVS-suite: A Simulator Supporting Visual Experimentation Design and Behavior Monitoring. In: *Proceedings of the 2009 Spring Simulation Multiconference.* San Diego, CA, USA : Society for Computer Simulation International, 2009 (SpringSim '09), 161:1–161:7

[53] Kolassa, Carsten ; Rendel, Holger ; Rumpe, Bernhard: Evaluation of Variability Concepts for Simulink in the Automotive Domain. In: *2015 48th Hawaii International Conference on System Sciences*, IEEE, 05.01.2015 - 08.01.2015. – ISBN 978–1–4799–7367–5, S. 5373–5382

[54] Koren, Yoram ; Gu, Xi ; Guo, Weihong: Reconfigurable manufacturing systems: Principles, design, and future trends. In: *Frontiers of Mechanical Engineering 13* (2018), S. 121–136. http://dx.doi.org/10.1007/s11465-018-0483-0. – DOI 10.1007/s11465–018–0483–0

[55] Kosuge, Kazuhiro ; Ishikawa, Jun: Task-oriented control of single-master multi-slave manipulator system. In: *Robotics and Autonomous Systems* 12 (1994), Nr. 1-2, S. 95–105. http://dx.doi.org/10.1016/0921-8890(94)90048-5. – DOI 10.1016/0921–8890(94)90048–5. – ISSN 09218890

[56] Kugelmann, Doris: *Forschungsberichte / IWB*. Bd. 127: *Aufgabenorientierte Offline-Programmierung von Industrierobotern: Zugl.: München, Techn. Univ., Diss., 1999*. München : Utz Wiss, 1999. – ISBN 389675615X

[57] Li, Wei ; Mani, Ramamurthy ; Mosterman, Pieter J.: Extensible Discrete-Event Simulation framework in SimEvents. In: *2016 Winter Simulation Conference (WSC)*, IEEE, 11.12.2016 - 14.12.2016. – ISBN 978–1–5090–4486–3, S. 943–954

[58] Litz, Lothar ; Frey, Georg: ÜBERSICHTSAUFSATZ · SURVEY PAPER: Methoden und Werkzeuge zum industriellen Steuerungsentwurf - Historie, Stand, Ausblick. In: *at - Automatisierungstechnik* 47 (1999), Nr. 4, S. 145–156. http://dx.doi.org/10.1524/auto.1999.47.4.145. – DOI 10.1524/auto.1999.47.4.145. – ISSN 0178–2312

[59] Lunze, Jan: *Künstliche Intelligenz für Ingenieure*. München and Wien : Oldenbourg, 1995. – ISBN 3–486–22306–2

[60] Lunze, Jan: *Automatisierungstechnik: Methoden für die Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme*. Oldenbourg Wissenschaftsverlag, 2003. – ISBN 3–486–27430–9

[61] Lunze, Jan: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. 10., aktual. Aufl. Berlin : Springer Vieweg, 2014 (Springer-Lehrbuch). – ISBN 978–3–642–53908–4

[62] Lüth, Tim ; Längle, Thomas: Task Description, Decomposition, and Allocation in a Distributed Autonomous Multi-Agent Robot System. (1994)

[63] Lüth, Tim ; Längle, Thomas: Multi-Agenten-Systeme in der Robotik und Artificial-Life. (1995)

[64] Lüth, Tim C.: *Technische Multi-Agenten-Systeme: Verteilte autonome Roboter- und Fertigungssysteme: Zugl.: Karlsruhe, Univ., Habil.-Schr., 1997*. München and Wien : Hanser, 1998. – ISBN 3–446–19468–1

[65] Makris, S. ; Michalos, G. ; Eytan, A. ; Chryssolouris, G.: Cooperating Robots for Reconfigurable Assembly Operations: Review and Challenges. In: *Procedia CIRP* 3 (2012), S. 346–351. http://dx.doi.org/10.1016/j.procir.2012.07.060. – DOI 10.1016/j.procir.2012.07.060. – ISSN 22128271

[66] Malakuti, Somayeh ; Schalkwyk, Pieter ; Boss, Birgit ; Sastry, Chellury ; Runkana, Venkat ; Lin, Shi-Wan ; Rix, Simon ; Green, Gavin ; Baechle, Kilian ; Nath, Shyam: Digital Twins for Industrial Applications. Definition, Business Values, Design Aspects, Standards and Use Cases. (2020), 02. https://www.researchgate.net/publication/339460951_Digital_Twins_for_Industrial_Applications_Definition_Business_Values_Design_Aspects_Standards_and_Use_Cases

[67] Maletzki, Gunnar: *Rapid Control Prototyping komplexer und flexibler Robotersteuerungen auf Basis des SBC-Ansatzes.* Uni. Rostock, Rostock University, Dissertation, 14.03.2014. http://www.cea-wismar.de/pubs/phd/dissertation-maletzki.pdf

[68] Maletzki, Gunnar ; Pawletta, Thorsten ; Dünow, Peter ; Manemann, P.: Simulationsmodellbasierte Steuerung einer Roboterzelle. In: *Frontiers in Simulation - Simulationstechnique 18th Symposium in Erlangen* (2005), S. 305–310

[69] Matarić, Maja J.: *The robotics primer.* Cambridge, Mass : MIT Press, 2007 (Intelligent robotics and autonomous agents series). http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=710638. – ISBN 978–0–262–63354–3

[70] Mataric, Maja J.: *The Basics of Robot Control.* http://agents.sci.brooklyn.cuny.edu/corc3303/papers/e1-mataric-robot-control.pdf. Version: 2010

[71] Meissner, Alexander: *Aufgabenorientierte Programmierung von Bewegungsbahnen für Grossroboter mit redundanter Gelenkarmkinematik.* Jost-Jetter-Verlag, 2000. – ISBN 978–3931388447

[72] Mittal, Saurabh ; Tolk, Andreas ; Pyles, Andrew ; van Balen, Nicolas ; Bergollo, Kevin: Digital Twin Modeling, Co-Simulation and Cyber Use-Case Inclusion Methodology for IOT Systems. In: *2019 Winter Simulation Conference (WSC)*, IEEE, 08.12.2019 - 11.12.2019. – ISBN 978–1–7281–3283–9, S. 2653–2664

[73] Moallemi, Mohammad: *Real-Time and Embedded Systems Development based on Discrete Event Modeling and Simulation.* Ottawa, Carleton University, Dissertation, Sep 2011. http://cell-devs.sce.carleton.ca/publications/2011/Moa11

[74] Moallemi, Mohammad ; Wainer, Gabriel: Designing an interface for real-time and embedded DEVS. In: McGraw, Robert (Hrsg.) ; Imsand, Eric (Hrsg.) ; Chinni, Michael J. (Hrsg.): *Proceedings of the 2010 Spring Simulation Multiconference on - SpringSim '10.* New York, New York, USA : ACM Press, 2010. – ISBN 9781450300698, 1

[75] Moler, Cleve: *Lehrbücher von Cleve Moler.* https://de.mathworks.com/moler.html. Version: 24.02.2020

[76] Open Source Robotics Foundation: *Robot Operating System.* http://www.ros.org/

[77] Otto, Johannes: *Entwicklung einer MATLAB-Visualisierungs-Toolbox für kooperierende Robotersysteme.* Wismar, Hochschule Wismar, Bachelor-Thesis, 2011

[78] Otto, Johannes ; Schwatinski, Tobias ; Pawletta, Thorsten: *KUKA-KAWASAKI-Visualization Toolbox for Matlab.* `http://www.cea-wismar.de/tbx/MatlabKK_Robotic-and-Visualization_Tbx/MatlabKK_Robotic_Visualization_Tbx.html`. Version: 2011

[79] Parker, Lynne E.: Path Planning and Motion Coordination in Multiple Mobile Robot Teams. In: Meyers, Robert A. (Hrsg.): *Encyclopedia of Complexity and System Science*, Springer, 2009

[80] Pawletta, Sven: Erweiterung einer wissenschaftlich-technischen berechnungs- und Visualisierungsumgebung zu einer Entwicklungsumgebung für parallele Applikationen. (2000), Nr. 7, S. 1–146

[81] Pawletta, T. ; Pascheka, D. ; Schmidt, A.: System Entity Structure Ontology Toolbox for MATLAB/Simulink: Used for Variant Modelling. In: *IFAC-PapersOnLine* 48 (2015), Nr. 1, S. 685–686. `http://dx.doi.org/10.1016/j.ifacol.2015.05.188`. – DOI 10.1016/j.ifacol.2015.05.188. – ISSN 24058963

[82] Pawletta, Thorsten ; Pascheka, Daniel. ; Schmidt, Artur ; Freymann, Birger, Deatcu, Christina ; Schwatinski, Tobias: *SES Toolbox for MATLAB / Simulink.* `http://www.cea-wismar.de/tbx/SES_Tbx/sesToolboxMain.html`

[83] Pawletta, Thorsten ; Pascheka, Daniel ; Schmidt, Artur ; Pawletta, Sven: Ontology-Assisted System Modeling and Simulation within MATLAB/Simulink. In: *SNE Simulation Notes Europe* 24 (2014), Nr. 2, S. 59–68. `http://dx.doi.org/10.11128/sne.24.tn.10241`. – DOI 10.11128/sne.24.tn.10241. – ISSN 2305–9974

[84] Pawletta, Thorsten ; Pascheka, Daniel ; Schmidt, Artur ; Pawletta, Sven: Ontology-Assisted System Modeling and Simulation within MATLAB/Simulink. In: *SNE Simulation Notes Europe* (2014), Nr. 24(2)-8/2014, 59–68. `http://dx.doi.org/10.11128/sne.24.tn.102241`. – DOI 10.11128/sne.24.tn.102241. – ISSN 2305–9974

[85] Pawletta, Thorsten ; Pawletta, Sven ; Maletzki, Gunnar: Integrated Modeling, Simulation and Operation of High Flexible Discrete Event Controls. In: *MATHMOD 09 : 6th Vienna Conference on Mathematical Modelling* (2010). `https://www.researchgate.net/publication/262425141_INTEGRATED_MODELING_SIMULATION_AND_OPERATION_OF_HIGH_FLEXIBLE_DISCRETE_EVENT_CONTROLS`

[86] Pfetzing, Karl ; Rohde, Adolf: *Ibo-Schriftenreihe*. Bd. 2: *Ganzheitliches Projektmanagement*. 5., überarb. Aufl. Gießen : Schmidt, 2014. – ISBN 978–3–921313–90–9

[87] Pichler, Franz ; Schwaertzel, Heinz: *CAST Methods in Modelling: Computer Aided Systems Theory for the Design of Intelligent Machines.* Springer-Verlag Berlin Heidelberg, 1992. – ISBN 978–3–642–95680–5

[88] Praehofer, H. ; Pree, D.: Visual Modeling of Devs-Based Multiformalism Systems Based on Higraphs. In: *Proceedings of 1993 Winter Simulation Conference - (WSC '93)*, IEEE, 1993. – ISBN 0–7803–1381–X, S. 595–603

[89] Quesnel, Gauthier ; Duboz, Raphaël ; Ramat, Éric ; Traoré, Mamadou K.: VLE: A Multimodeling and Simulation Environment. In: *Proceedings of the 2007 Summer Computer Simulation Conference*. San Diego, CA, USA : Society for Computer Simulation International, 2007 (SCSC '07). – ISBN 1–56555–316–0, 367–374

[90] Risco-Martín, José L. ; Mittal, Saurabh ; Fabero, Juan C. ; Malagón, Pedro ; Ayala, José L.: Real-time Hardware/Software Co-design Using Devs-based Transparent M&S Framework. In: *SummerSim-SCSC 2016 July 24-27 Montreal, Quebec, Canada* (2016), 45:1–45:8. http://dl.acm.org/citation.cfm?id=3015574.3015619

[91] Rokossa, Dirk: *Prozeßorientierte Offline-Programmierung von Industrierobotern: Univ., Diss.–Dortmund, 1999*. Als Ms. gedr. Aachen : Shaker, 2000 (Berichte aus der Automatisierungstechnik). – ISBN 3826569458

[92] Rozenblit, Jerzy W.: Experimental Frame Specification Methodology for Hierarchical Simulation Modeling. In: *International Journal of General Systems* 19 (1991), Nr. 3, S. 317–336. http://dx.doi.org/10.1080/03081079108935180. – DOI 10.1080/03081079108935180. – ISSN 0308–1079

[93] Russell, Stuart J. ; Norvig, Peter: *Künstliche Intelligenz: Ein moderner Ansatz*. 3., aktualisierte Aufl. München : Pearson, 2012 (Always learning). http://lib.myilibrary.com/detail.asp?id=404935. – ISBN 9783868940985

[94] Sarjoughian, Hessam S. ; Elamvazhuthi, Vignesh: CoSMoS: a visual environment for component-based modeling, experimental design, and simulation. In: Dalle, Olivier (Hrsg.) ; Wainer, Gabriel (Hrsg.) ; Stea, Giovanni (Hrsg.) ; Perrone, L. F. (Hrsg.): *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, ICST, 02.03.2009 - 06.03.2009. – ISBN 978–963–9799–45–5

[95] Sarjoughian, Hessam S. ; Gholami, Soroosh: Action-level real-time DEVS modeling and simulation. In: *SIMULATION* 91 (2015), Nr. 10, S. 869–887. http://dx.doi.org/10.1177/0037549715604720. – DOI 10.1177/0037549715604720. – ISSN 0037–5497

[96] Schalkwyk, Pieter ; Malakuti, Somayeh ; Lin, Shi-Wan: A Short Introduction to Digital Twins. (2019), 11. https://www.researchgate.net/publication/337674017_A_Short_Introduction_to_Digital_Twins

[97] Schmidt, Artur: *Variantenmanagement in der Modellbildung und Simulation unter Verwendung des SES/MB Frameworks*. Rostock, Universität Rostock, Dissertation, 2019

[98] Schmidt, Artur ; Christern, Michael: *Entwicklung einer KUKA-Kawasaki-Robotik-Toolbox mit MATLAB und Scilab*. Wismar, Hochschule Wismar, Bachelor-Thesis, 2009

[99] Schmidt, Artur ; Durak, Umut ; Rasch, Christoph ; Pawletta, Thorsten: Model-Based Testing Approach for MATLAB/Simulink using System Entity Structure and Experimental Frames. In: *2015 Spring Simulation Multi-Conference (SpringSim'15), Symposium on Theory of Modeling and Simulation (TMS/DEVS), Alexandria, VA* (2015), S. 828–835

[100] Scholz, Peter: *Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung.* Berlin : Springer, 2005 (Xpert.press). – ISBN 3540275223

[101] Schreiber, Günter: *Steuerung für redundante Robotersysteme: Benutzer- und aufgabenorientierte Verwendung der Redundanz*

[102] Schwatinski, Tobias ; Pawletta, Thorsten ; Pawletta, Sven: Flexible Task Oriented Robot Controls Using the System Entity Structure and Model Base Approach. In: *SNE Simulation Notes Europe* 22 (2012), Nr. 2, S. 107–114. http://dx.doi.org/10.11128/sne.22.tn.10135. – DOI 10.11128/sne.22.tn.10135. – ISSN 2305–9974

[103] Schwatinski, Tobias ; Pawletta, Thorsten ; Pawletta, Sven ; Kaiser, Christian: Simulation-based development and operation of controls on the basis of the DEVS formalism. In: *Proc. of The 7th EUROSIM 2010 Congress* 2 (2010), Nr. Volume 2

[104] Scilab Enterprises: *Scilab*. https://www.scilab.org/

[105] Senfelds, Armands: Analysis of motion modelling approaches for industrial robot applications. In: *2019 IEEE 7th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, IEEE, 15.11.2019 - 16.11.2019. – ISBN 978–1–7281–6730–5, S. 1–4

[106] Shannon, Robert E.: *Systems simulation: The art and science.* Englewood Cliffs, NJ : Prentice-Hall, 1975. – ISBN 978–0138818395

[107] Siciliano, Bruno (Hrsg.): *Springer handbook of robotics.* Berlin u.a. : Springer, 2008. – ISBN 9783540239574

[108] Silano, Giuseppe ; Oppido, Pasquale ; Iannelli, Luigi: Software-in-the-loop simulation for improving flight control system design: a quadrotor case study. In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, S. 466–471

[109] Simmons, R. ; Apfelbaum, D.: A task description language for robot control. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, IEEE, 13-17 Oct. 1998. – ISBN 0–7803–4465–0, S. 1931–1937

[110] Song, Hae S. ; Kim, Tag G.: Application of Real-Time DEVS to Analysis of Safety-Critical Embedded Control Systems: Railroad Crossing Control Example. In: *SIMULATION* 81 (2005), Nr. 2, S. 119–136. http://dx.doi.org/10.1177/0037549705052229. – DOI 10.1177/0037549705052229. – ISSN 0037–5497

[111] Song, Hae S. ; Kim, Tag G.: Devs Diagram Revised: A Structured Approach for Devs Modeling. (2010)

[112] Steffen Ritter: *Autonome Multi-Agenten-Systeme in der Industrie: Informatik Master Fakultät Elektrotechnik & Informationstechnik*, Offenbach, Diss., 30.04.2015. https://opus.hs-offenburg.de/files/963/Master-Thesis.pdf

[113] Stenmark, Maj ; Malec, Jacek: Knowledge-Based Industrial Robotics. In: Manfred Jaeger (Hrsg.) ; Dyhre Nielsen, Thomas (Hrsg.) ; Paolo Viappiani (Hrsg.): *Frontiers in Artificial Intelligence and Applications* Bd. 257. Netherlands : IOS Press, 2013. – ISBN 978–1–61499–330–8, S. 265–274

[114] Tanenbaum, Andrew S. ; Bos, Herbert: *Moderne Betriebssysteme*. 4., aktualisierte Auflage. Hallbergmoos : Pearson, 2016 (Always learning). `http://lib.myilibrary.com?id=927255`. – ISBN 9783868942705

[115] Tang, Li ; Koren, Y. ; Yip-Hoi, D. ; Wang, Wencai: Computer-Aided Reconfiguration Planning: An Artificial Intelligence-Based Approach. In: *Journal of Computing & Information Science in Engineering (JCISE)* 6 (2006), S. 230–240

[116] Tolk, Andreas (Hrsg.) ; Durak, Umut (Hrsg.): *Simulation als epistemologische Grundlage für intelligente Roboter.* Ulm, Germany : Workshop der ASIM/GI Fachgruppen STS und GMMS, 2017 `https://elib.dlr.de/112667/1/TolkDurakASIM2017.pdf`

[117] Traoré, Mamadou K.: SimStudio: a Next Generation Modeling and Simulation Framework. In: Molnár, Sándor (Hrsg.) ; Heath, John (Hrsg.): *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*, ICST, 03.03.2008 - 07.03.2007. – ISBN 978–963–9799–23–3

[118] Traoré, Mamadou K. ; Muzy, Alexandre: Capturing the dual relationship between simulation models and their context. In: *Simulation Modelling Practice and Theory* 14 (2006), Nr. 2, S. 126–142. `http://dx.doi.org/10.1016/j.simpat.2005.03.002`. – DOI 10.1016/j.simpat.2005.03.002. – ISSN 1569190X

[119] van Harmelen, Frank (Hrsg.) ; Lifschitz, Vladimir (Hrsg.) ; Porter, Bruce (Hrsg.): *Handbook of knowledge representation.* 1st ed. Amsterdam and Boston : Elsevier, 2008 (Foundations of artificial intelligence). `http://www.sciencedirect.com/science/publication?issn=15746526&volume=3`. – ISBN 978–0–444–52211–5

[120] van Mierlo, Simon ; van Tendeloo, Yentl ; Vangheluwe, Hans: Debugging Parallel DEVS. In: *SIMULATION* 93 (2017), Nr. 4, S. 285–306. `http://dx.doi.org/10.1177/0037549716658360`. – DOI 10.1177/0037549716658360. – ISSN 0037–5497

[121] VISUAL COMPONENTS: *VISUAL COMPONENTS*. `https://www.visualcomponents.com/de/`

[122] Wainer, Gabriel: CD++: a toolkit to develop DEVS models. In: *Software: Practice and Experience* 32 (2002), Nr. 13, S. 1261–1306. `http://dx.doi.org/10.1002/spe.482`. – DOI 10.1002/spe.482. – ISSN 0038–0644

[123] Wang, Zongyan: Digital Twin Technology. Version: 2020. `http://dx.doi.org/10.5772/intechopen.80974`. In: Bányai, Tamás (Hrsg.) ; Petrilloand Fabio De Felice, Antonella (Hrsg.): *Industry 4.0 - Impact on Intelligent Logistics and Manufacturing*. IntechOpen, 2020. – DOI 10.5772/intechopen.80974. – ISBN 978–953–51–6996–3, S. 95 – 114

[124] Weber, Wolfgang: *Industrieroboter: Methoden der Steuerung und Regelung; mit 33 Übungsaufgaben sowie einer begleitenden Internetseite.* 2., neu bearb. Aufl. München : Fachbuchverl. Leipzig im Carl-Hanser-Verl., 2009. – ISBN 9783446410312

[125] Weller, W.: *Automatisierungstechnik im Überblick: Was ist, was kann Automatisierungstechnik?* 1. Aufl. s.l. : Beuth Verlag GmbH, 2008 (Beuth Wissen). http://gbv.eblib.com/patron/FullRecord.aspx?p=2031573. – ISBN 978–3–410–16760–0

[126] Willow Garage, Inc.: *Open Source Robotics Foundation.* http://www.willowgarage.com/blog/2012/04/16/open-source-robotics-foundation. Version: 04.16.2012

[127] Yan, Zhi ; Jouandeau, Nicolas ; Cherif, Arab A.: A Survey and Analysis of Multi-Robot Coordination. In: *International Journal of Advanced Robotic Systems* 10 (2013), Nr. 12, S. 399. http://dx.doi.org/10.5772/57313. – DOI 10.5772/57313. – ISSN 1729–8814

[128] Yilmaz, Levent ; Ören, Tuncer I.: Dynamic model updating in simulation with multimodels: A taxonomy and a generic agent-based architecture. 36/4 (2004), 3–8. http://site.uottawa.ca/~oren/pubs/pubs-2004-04-SCSC-MM.pdf

[129] Zander, Hans-Joachim: *Steuerung ereignisdiskreter Prozesse: Neuartige Methoden zur Prozessbeschreibung und zum Entwurf von Steueralgorithmen.* Wiesbaden : Springer Vieweg, 2015. http://dx.doi.org/10.1007/978-3-658-01382-0. http://dx.doi.org/10.1007/978-3-658-01382-0. – ISBN 9783658013813

[130] Zeigler, B. P. ; Kim, J.: Extending the DEVS-Scheme knowledge-based simulation environment for real-time event-based control. In: *IEEE Transactions on Robotics and Automation* 9 (1993), Nr. 3, S. 351–356. http://dx.doi.org/10.1109/70.240207. – DOI 10.1109/70.240207

[131] Zeigler, B. P. ; Praehofer, H. ; Kim, T.G: *Theory of Modeling and Simulation: Integrating Discrete Event and Continous Complex Dynamic Systems.* 2nd ed Elsevier Academic Press, 2000

[132] Zeigler, Bernard P.: *Theory of modelling and simulation.* New York : Wiley, 1976 (A Wiley-Interscience publication). – ISBN 9780471981527

[133] Zeigler, Bernard P.: *Multifacetted modelling and discrete event simulation.* London : Acad. Pr, 1984. – ISBN 0–12–778450–0

[134] Zeigler, Bernard P. ; Hammonds, Phillip E.: *Modeling & simulation-based data engineering: Introducing pragmatics into ontologies for net-centric information exchange.* Amsterdam, Boston : Elsevier Academic Press, 2007. – ISBN 9780123725158

[135] Zeigler, Bernard P. ; Muzu, Alexandre ; Kofman, Ernesto: *Theory of modeling and simulation: Discrete event and iterative system computational foundations.* 3rd. London, United Kingdom and San Diego, CA, United States : Academic Press, 2018. – ISBN 9780128133705

[136] Zeigler, Bernard P. ; Sarjoughian, Hessam S.: *Guide to Modeling and Simulation of Systems of Systems*. London : Springer London, 2013 (Simulation Foundations, Methods and Applications). – ISBN 9780857298645

[137] Zinoviev, Dmitry: Mapping DEVS Models onto UML Models. In: *CoRR* abs/cs/0508128 (2005)

Abbildungsverzeichnis

2.1 Einflussgrößen und Zielgröße der Steuerungsentwicklung 12
2.2 Mögliche Varianten einer Steuerungsarchitektur und zugehöriges Steuerungsparadigma schematisch entworfen auf Basis von Mataric [70] 13
2.3 Varianten von Steuerungsarchitekturen in Anlehnung an Burth [3] 14
2.4 Zusammenhang zwischen Steuerungsparadigmen nach Mataric [70] und Aktivitätsniveaus nach Burth [3] 15
2.5 Linke Seite: Hierarchische Struktur von Robotersteuerungen nach Lunze [61] Rechte Seite: Allgemeine Struktur einer Supervisory-Control nach Groover [42] 16
2.6 Ausgewählte Roboterhersteller und ihre Programmiersprachen 17
2.7 Steuerungsentwicklung nach dem Client-Server Modell 18
2.8 Typische Methoden der Roboterprogrammierung 19
2.9 Vergleich zwischen realer und virtueller Prozessumgebung 23
2.10 Realisierung einer Automatisierungslösung nach dem V-Modell gemäß Maletzki [67] 24
2.11 Angepasstes V-Model zur Robotersteuerungsentwicklung von Maletzki [67] 25
2.12 Verbindung von Systemsimulation, Hard- und Software in the Loop nach Abel [1] 26
2.13 Schematische Darstellung der SBC-Vorgehensweise für die Steuerungsentwicklung 28
2.14 De-/Komposition eines Problems in mehrere Teilaufgaben 29
2.15 Schichten einer aufgabenorientierten Steuerung nach Weber [124] 30
2.16 Umsetzung aufgabenorientierter Steuerungen im SBC-Ansatz 31
2.17 Implizite und explizite Kommunikation von MRS nach Behnke [8] 34
2.18 Zusammenhang von VS, MRS und MAS 34
2.19 Transportproblem mit einen SRS 35
2.20 Schematische Darstellung des Transportproblems für die Interaktionsklassen eines MRS in Anlehnung an die Klassifikation in Lüth [63] 36

3.1 Struktur eines DEVS-Computermodells 41
3.2 Dynamik eines Atomic-Classic-DEVS nach Zeigler [131] 42
3.3 Dynamik eines Atomic-Parallel-DEVS bei zeitgleichem internen und externen Ereignis nach Zeigler [131] 46
3.4 Schematische Darstellung eines Atomic-(P)DEVS mittels DEVS-Diagramm nach Song [111] einschließlich eingeführter Erweiterung *Aktivitäten* 48
3.5 Spezifikation eines Atomic-Classic-DEVS namens PROC mittels DEVS-Diagramm 50
3.6 Dynamik eines Atomic-Real-Time-DEVS nach Zeigler [131] 53

3.7 Vorgehensmodell der RT-DEVS/Corba-Umgebung nach Cho [18] 56
3.8 Problematische Umsetzung eines Füllstandssensors mittels RT-DEVS-Formalismus 57
3.9 Simulationsalgorithmus des ALRT-DEVS-Ansatzes von Sarjoughian und Gholami [95] 59
3.10 Schematische Darstellung eines RTE-RDEVS-Modells und Einordnung in den SBC-Ansatz 62
3.11 Schematische Darstellung elementarer Komponenten des Transparent-DEVS-Frameworks nach Risco Martin [90] 63
3.12 Modell einer Produktionsanlage bestehend aus Generator (GEN), Prozessor (PROC) und Transducer (TRA) nach Risco-Martin [90] 64
3.13 Dynamik eines Atomic-PDEVS-RCP basierend auf Schwatinski [103] . . . 66
3.14 Atomic-PDEVS-RCP-Spezifikation einer Echtzeituhr basierend auf Schwatinski [103] 66

4.1 Gewünschte durchgängige Steuerungsentwicklung mit (P)DEVS 70
4.2 Kombination eines Atomic-PDEVS mit einer Menge FNSS zur Umsetzung von Prozessinteraktionen in PDEVS-RCP-V2 72
4.3 Spezifikation der Real-Time-Clock (RTC) nach PDEVS-RCP-V2 74
4.4 Anwendungsbeispiel nach SBC-Ansatz 75
4.5 Vordefinierte Positionen home und home2 76
4.6 Erweitertes DEVS-Diagramm INTF in der Automatisierungsphase 76
4.7 Erweitertes DEVS-Diagramm GEN 77
4.8 Erweitertes DEVS-Diagramm PROC 77
4.9 Erweitertes DEVS-Diagramm TRA 78
4.10 Erweitertes DEVS-Diagramm INTF im operativen Betrieb 79
4.11 Beispiel im Kontext von SBC und TOC 80
4.12 Varianten zur Zuordnung von Aufgaben (Ti) zu Robotern (Ri) durch die CM-Ebene 81
4.13 Lösungsansatz für Interaktionsklasse 1 und 2 82
4.14 Lösungsvarianten auf PM-Ebene 83
4.15 Lösungsansatz für Interaktionsklasse 3 83
4.16 Die Aufgabe PickPlaceMutex 84
4.17 Lösungsansatz für Interaktionsklasse 4 85
4.18 Varianten zur Umsetzung des RT auf PM-Ebene 87
4.19 Die komponierte Aufgabe JoinRTIf 87
4.20 Die komponierte Aufgabe PickPlaceSync 88
4.21 Lösungsansatz für Interaktionsklasse 5 88
4.22 Erster Lösungsansatz für Interaktionsklasse 6 (Werkzeugtausch) 89
4.23 Komposition der Aufgabe Reconfig 90
4.24 Zweiter Lösungsansatz für Interaktionsklasse 6 (temporärer Roboter) . . . 90
4.25 Komposition der Aufgabe SwapRole 91

5.1 Beispiel einer SES 94
5.2 Beispiel einer PES und einer FPES zur SES in Abbildung 5.1 96
5.3 Erweitertes SES/MB-Framework zum automatisierten Experimentieren nach Schmidt [97], angepasst auf die Problematik flexibler Steuerungen . . 97
5.4 Änderung der Steuerungskonfiguration einer nach dem SBC-Ansatz strukturierten flexiblen Steuerung 98

5.5 Struktur eines SM mit einer Steuerungskonfiguration (MUS) und dem dazugehörigen Kontext (EF) 99

6.1 Beispiel zur Visualisierung mehrerer Roboter mittels RCV-Toolbox [39] . . 103
6.2 Grafisch unterstütztes Debuggen einer PDEVS-basierten Robotersteuerung 104
6.3 Graphische Nutzerschnittstelle der SES Toolbox for MATLAB/Simulink . . 105
6.4 Softwaretechnische Umsetzung des Erweiterten SES/MB-Frameworks zur Realisierung flexibler Multi-Robotersteuerungen 107
6.5 Erweitertes DEVS-Diagramm der Aufgabe IdPrt 109
6.6 Erweitertes DEVS-Diagramm der Aufgabe Move 109
6.7 Erweitertes DEVS-Diagramm einer allgemeinen Aufgabe 110
6.8 Erweitertes DEVS-Diagramm der Aufgabe no operation NoOp 110
6.9 Erweitertes DEVS-Diagramm der Roboterkomponente R 112
6.10 Erweitertes DEVS-Diagramm der Interfacekomponente INTF 113
6.11 Erweitertes DEVS-Diagramm der Roboterkomponente R 114
6.12 Erweitertes DEVS-Diagramm der Komponente Roboterteam RT 115
6.13 Erweitertes DEVS-Diagramm der Komponente Roboter R 118
6.14 Erweitertes DEVS-Diagramm der Komponente Roboterteam RT 119
6.15 Zusammenspiel der Aufgaben Sleep und WakeUp mit Roboter- und Roboterteam-Komponenten 120
6.16 Lösungsansatz für Interaktionsklasse 6 mit einer SES 121
6.17 Steuerungskonfiguration mit zwei Robotern für die Bauteile A, B und C . . 122
6.18 Steuerungskonfiguration mit Spezialroboter R3 für Bauteil D 123
6.19 Struktur des Experimental-Frame 123
6.20 Erweitertes DEVS-Diagramm der Komponente GEN 124
6.21 Erweitertes DEVS-Diagramm der Komponente TRA 124
6.22 Erweitertes DEVS-Diagramm der Komponente ACC 125

A.1 Erweiterte DEVS-Diagramme zu den Varianten einer RTC mittels PDEVS-RCP-V2 154
B.2 Erweitertes DEVS-Diagramm GEN 155
B.3 Erweitertes DEVS-Diagramm PROC 156
B.4 Erweitertes DEVS-Diagramm TRA 157
B.5 Erweitertes DEVS-Diagramm INTF (Automatisierungsphase) 157
B.6 Erweitertes DEVS-Diagramm INTF (operativer Betrieb) 158
C.7 Interaktionsklasse 6 ohne komponierte Aufgaben 160
D.8 Beispiele zu den Axiomen der SES 161

Tabellenverzeichnis

2.1 Wesentliche Merkmale der Interaktionsklassen 37

3.1 Atomic-Classic-DEVS 42
3.2 Atomic-Classic-DEVS mit Ports 43
3.3 Coupled-DEVS auch genannt DEVS-Network (DEVN) 43
3.4 Nachrichtenprotokoll der Classic-DEVS Simulationsalgorithmen 44
3.5 Wesentliche Eigenschaften des PDEVS-Ansatzes 46
3.6 Atomic-Parallel-DEVS (PDEVS) 46
3.8 Liste DEVS-basierter Simulatoren (unvollständig) 47
3.7 Coupled-Parallel-DEVS (PDEVN) 47
3.9 DEVS-Diagramm-Notation und eingeführte Erweiterungen 49
3.10 PDEVS-Mengennotation eines PROC 51
3.11 Coupled-Real-Time-DEVS, auch genannt Real-Time-DEVS-Network (RT-DEVN) 52
3.12 Atomic-Real-Time-DEVS (RT-DEVS) 52
3.13 Vergleich der Spezifikation von Classic-DEVS-Modellen nach Zeigler et al. [131] mit RT-DEVS-Modellen nach Hong et al. [48] 55
3.14 Atomic-Action-Level-Real-Time-DEVS nach Sarjoughian und Gholami [95] 58
3.15 Coupled-RTE-PDEVS (RTE-PDEVN) nach Moallemi [74] 61
3.16 Real-Time-DEVS-Treiber (RT-DM) nach Moallemi [74] 61
3.17 Atomic-Transparent-DEVS (TR-DEVS) 64
3.18 Atomic PDEVS RCP 65
3.19 Voraussetzungen der Echtzeitsimulation eines Atomic-PDEVS-RCP 66

4.1 Atomic-PDEVS-RCP-V2 73
4.2 PDEVS-Mengennotation einer RTC 74

5.1 Axiome der SES 95

6.1 Ausgewählte Befehle der RCV-Toolbox 103
6.2 Kodierung und Parameter der Aufgaben für Interaktionsklasse 1 und 2 . . 108
6.3 Parameter und Kodierung der Aufgaben für Interaktionsklasse 3 114
6.4 Parameter und Kodierung der Aufgaben für Interaktionsklasse 6 117

B.1 PDEVS-Mengennotation eines GEN 155
B.2 PDEVS-Mengennotation eines PROC 156
B.3 PDEVS-Mengennotation eines TRA 157
B.4 PDEVS-Mengennotation eines INTF (Automatisierungspahse) 158
B.5 PDEVS-Mengennotation eines INTF (operativer Betrieb) 159

E.6 Liste der entwickelten PDEVS-Modelle zu Kapitel 6 162
E.7 Liste der entwickelten PDEVS-Modelle zu Kapitel 6 162

Listings

3.1 DEVS-Simulator nach [131] 45
3.2 DEVS-Koordinator nach [131] 45
3.3 star-Message des RT-DEVS-Simulators nach [48] 54
3.4 x-Message des RT-DEVS-Simulator nach [48] 54
3.5 Algorithmus eines RT-DEVS-Simulators nach [48] 54

6.1 Pseudocode der Execution-Control (EC) 125

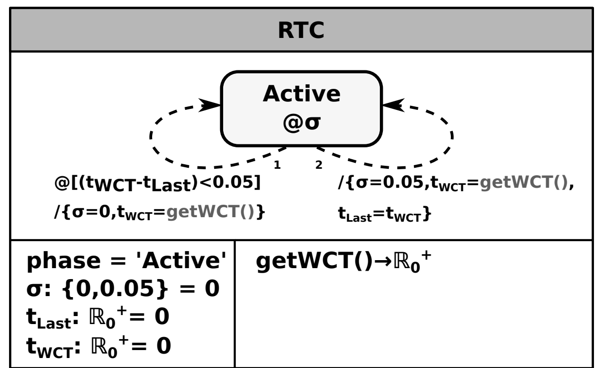
G.1 Atomic-PDEVS-Modell der Aufgabe GTask 162
G.2 Atomic-PDEVS-Modell der Aufgabe IdPrt 163
G.3 Atomic-PDEVS-Modell der Aufgabe NoOp 164
G.4 Atomic-PDEVS-Modell der Komponente R 165
G.5 Atomic-PDEVS-Modell der Komponente RT 167
G.6 Atomic-PDEVS-Modell der Komponente INTF 172
G.7 Atomic-PDEVS-Modell der Komponente RTC 173
G.8 Coupled-PDEVS RM zur Interaktionsklasse 0 174
G.9 Coupled-PDEVS RM zur Interaktionsklasse 1 und 2 175
G.10 Coupled-PDEVS RM zur Interaktionsklasse 3 176
G.11 Coupled-PDEVS RM zur Interaktionsklasse 4 178

Anhang

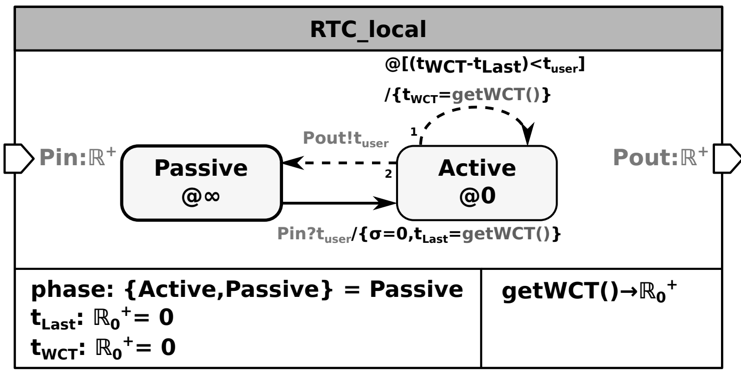
A Diskussion zur Umsetzung der Komponente RTC

Um mit der Prozessumgebung zu interagieren ist im SBC-Ansatz ein Interfacemodell vorgesehen. Dieses bildet die Schnittstelle zwischen dem Simulationsmodell und der zu steuernden Prozessumgebung ab. Eine Wechselwirkung mit der Prozessumgebung ist in der Regel zeitbehaftet. Hierbei wird häufig ein zulässigen Zeitinterval definiert, in welchem die Prozessumgebung zu reagieren hat. Wird das Zeitinterval ohne eine Antwort der Prozessumgebung verlassen, kann eine Störung aufgetreten sein und eine Korrektur des Steuerungsverhaltens kann erforderlich werden. Weiterhin soll der Übergang von der Entwurfs- zur Automatisierungsphase für den Entwickler durch einen durchgängigen Entwicklungsprozess vereinfacht werden. Hierfür muss sich das Simulationsmodell der Entwurfs-, um die neuen Anforderungen der Automatisierungsphase erweitern lassen. Aufgrund wohldefinierter Algorithmen soll Simulationsmodell in Entwurfsphase auf Basis des PDEVS-Formalismuses entwickelt werden. Folglich arbeiten die Simulationsalgorithmen auf Basis von virtueller Zeitfortschaltung. Wie in Abschnitt 2.3.3 disktutiert wird, ist es erforderlich, dass die entwickelten Simulationsmodelle in der Automatisierungsphase schrittweise um eine Prozessanbindung erweitert werden. Um Fehlertolerante Steuerungen realisieren zu können muss die virtuelle Zeitfortschaltung Echtzeitsynchronisiert werden. Da eine Anpassung der Simulationsalgorithmen nicht in Frage kommt, kann diese Synchronisation ausschließlich auf Modellebene erfolgen. Hierfür würde eine die entwicklung einer atomaren DEVS Komponente Real-Time-Clock (RTC) vorgeschlagen. In Abbildung A.1 werden zwei mögliche Varianten zur Umsetzung einer Echtzeituhr vorgestellt. In Teilbild (a) wird eine globale Echtzeituhr auf Basis der erweiterten DEVS-Diagramm-Notation gezeigt. Diese synchronisiert die virtuelle und die reale Zeitfortschaltung permanent für alle Modellkomponenten. Eine Abwandlung der grundlegenden Idee zeigt Teilbild (b). Hierbei zu sehen ist eine lokale Echtzeituhr. Diese synchronisiert die virtuelle Zeitfortschaltung des Simulators nicht mit der realen Zeitfortschaltung. Außerdem verfügt die lokale Echtzeituhr über Schnittstellen zur ereignisbasierten Kommunikation mit anderen Modellkomponenten. Die grundlegende Funktionalität ist: Empfangen einen Ereigniswertes $t_{user} \in \mathbb{R}^+$ über den Eingangsport Pin und Wechsel in den Zustand Aktive. Durch das Einplanen mehrerer interner Zeitereignisse erfolgt eine Überprüfung ob die mit t_{user} definierte Zeitdauer auf Basis realer Zeit verstrichen ist. Wenn dies der Fall ist, wird erneut in den Zustand Passive gewechselt und zuvor ein Ausgangsereignis mit den Wert t_{user} versendet. Dieses kann nunmehr zur Überprüfung eines zulässigen Zeitintervals verwendet werden. Die Realisierung der Echtzeituhren kann in beiden Fällen mittels PDEVS-RCP-V2-Formalismus erfolgen und erfordert keine Modifikation der Simulationsalgorithmen. Die lokale Echtzeituhr ermöglicht eine Mischung aus Echtzeitsynchronisation und einer As-Fast-As-Possible-Simulation

auf Basis von virtueller Zeitfortschaltung. Dies kann für den Entwicklungsprozess der Steuerung nützlich sein, falls mögliche Fehler erst im späteren Verlauf auftreten. Somit kann die lokale Echtzeituhr eine Unterstützung sein um eine Steuerung in der Entwicklung zu debuggen.



(a) Globale Echtzeituhr



(b) Lokale Echtzeituhr

Abbildung A.1: Erweiterte DEVS-Diagramme zu den Varianten einer RTC mittels PDEVS-RCP-V2

B Modellspezifikationen zum Abschnitt 4.2

In Abschnitt 4.2 wird mit Abbildung 4.4 ein Anwendungsbeispiel zur durchgängigen Steuerungsentwicklung präsentiert. Die entwickelten Modellkomponenten werden nachfolgend vergleichend als erweitertes DEVS-Diagramm und in DEVS-Mengennotation dargestellt.

Die nachfolgende Darstellungen zeigen den Vergleich zwischen erweiterter DEVS-Diagramm-Notation und einer mengentheoretischen Spezifikation der im Beispiel gezeigten atomaren DEVS-Modelle.

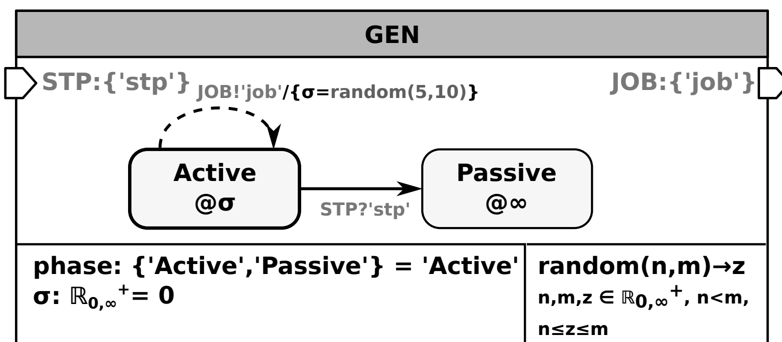


Abbildung B.2: Erweitertes DEVS-Diagramm GEN

Tabelle B.1: PDEVS-Mengennotation eines GEN

$\mathbf{GEN} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$	
$\mathbf{X}$	$\mathrm{X} = \{(\mathrm{STP}, \text{‘stp‘}) \mid \mathrm{STP} \in \mathrm{IPorts}\}$
$\mathbf{S}$	$\mathrm{S} = \{(\mathrm{phase}, \sigma) \mid \mathrm{phase} \in \{\text{‘Active‘}, \text{‘Passive‘}\}, \sigma \in \mathbb{R}^+_{0,\infty}\}$
	$\mathrm{s_{initial}} = (\text{‘Active‘}, 0)$
$\mathbf{Y}$	$\mathrm{Y} = \{(\mathrm{JOB}, \text{‘job‘}) \mid \mathrm{JOB} \in \mathrm{OPorts}\}$
$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}}((\mathrm{phase}, \sigma)) := (\text{‘Active‘}, \mathrm{random}(5, 10)) \mid$
	$\mathrm{random}(\mathrm{n}, \mathrm{m}) \rightarrow \mathrm{z} \quad \mathrm{n}, \mathrm{m}, \mathrm{z} \in \mathbb{R}^+_{0,\infty}, \mathrm{n} < \mathrm{m}, \mathrm{n} \leq \mathrm{z} \leq \mathrm{m}$
$\delta_{\mathbf{ext}}$	$\delta_{\mathrm{ext}}((\mathrm{phase}, \sigma), \mathrm{e}, (\mathrm{STP}, \text{‘stp‘})) := (\text{‘Passive‘}, \infty)$
$\delta_{\mathbf{con}}$	$\delta_{\mathrm{con}}((\mathrm{phase}, \sigma), \mathrm{e}, (\mathrm{STP}, \text{‘stp‘})) :=$
	$\delta_{\mathrm{ext}}(\delta_{\mathrm{int}}((\mathrm{phase}, \sigma)), \mathrm{e}, (\mathrm{STP}, \text{‘stp‘}))$
λ	$\lambda((\mathrm{phase}, \sigma)) := (\mathrm{JOB}, \text{‘job‘})$
$\mathbf{ta}$	$\mathrm{ta}((\mathrm{phase}, \sigma)) := \sigma$

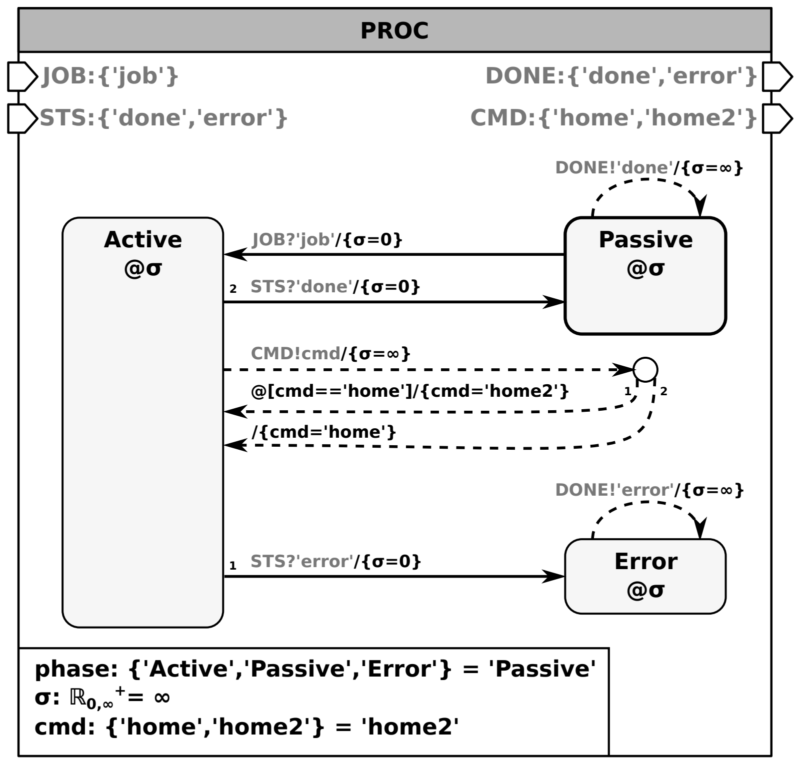


Abbildung B.3: Erweitertes DEVS-Diagramm PROC

Tabelle B.2: PDEVS-Mengennotation eines PROC

$\mathbf{PROC} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$	
X	$\mathrm{X} = \{((\mathrm{JOB}, \text{‘job‘})(\mathrm{STS}, \mathrm{sts})) \vert \mathrm{JOB}, \mathrm{STS} \in \mathrm{IPorts}, \mathrm{sts} \in \{\text{‘done‘}, \text{‘error‘}\}\}$
S	$\mathrm{S} = \{(\mathrm{phase}, \sigma, \mathrm{cmd}) \vert \mathrm{phase} \in \{\text{‘Active‘}, \text{‘Passive‘}, \text{‘Error‘}\},$ $\sigma \in \mathbb{R}^+_{0,\infty}, \mathrm{cmd} \in \{\text{‘home‘}, \text{‘home2‘}\}\}$ $\mathrm{s}_{\mathrm{initial}} = (\text{‘Passive‘}, \infty, \text{‘home2‘})$
Y	$\mathrm{Y} = \{((\mathrm{DONE}, \mathrm{dval})(\mathrm{CMD}, \mathrm{cmd})) \vert \mathrm{DONE}, \mathrm{CMD} \in \mathrm{OPorts},$ $\mathrm{dval} \in \{\text{‘done‘}, \text{‘error‘}\}, \mathrm{cmd} \in \{\text{‘home‘}, \text{‘home2‘}\}\}$

$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}}((\mathrm{phase}, \sigma, \mathrm{cmd})):$
	 if (‘Passive‘, 0, cmd) then: ((‘Passive‘, ∞, cmd))
	 if (‘Error‘, 0, cmd) then: ((‘Error‘, ∞, cmd))
	 if (‘Active‘, 0, ‘home‘) then: ((‘Active‘, ∞, ‘home2‘))
	 if (‘Active‘, 0, ‘home2‘) then: ((‘Active‘, ∞, ‘home‘))

$\delta_{\mathbf{ext}}$	$\delta_{\mathrm{ext}}((\mathrm{phase}, \sigma, \mathrm{cmd}), \mathrm{e}, ((\mathrm{JOB}, \text{‘job‘})(\mathrm{STS}, \mathrm{sts}))):$
	 if (‘Passive‘, ∞, cmd)&(JOB, ‘job‘) then: (‘Active‘, 0, cmd)
	 if (‘Active‘, σ, cmd)&(STS, ‘error‘) then: (‘Error‘, 0, cmd)
	 if (‘Active‘, σ, cmd)&(STS, ‘done‘) then: (‘Passive‘, 0, cmd)

$\delta_{\mathbf{con}}$	$\delta_{\mathrm{con}}((\mathrm{phase}, \sigma, \mathrm{cmd}), \mathrm{e}, ((\mathrm{JOB}, \text{‘job‘})(\mathrm{STS}, \mathrm{sts}))) :=$ $\delta_{\mathrm{ext}}(\delta_{\mathrm{int}}((\mathrm{phase}, \sigma, \mathrm{cmd})), \mathrm{e}, ((\mathrm{JOB}, \text{‘job‘})(\mathrm{STS}, \mathrm{sts})))$

λ	$\lambda((\mathrm{phase}, \sigma)):$
	 if (‘Passive‘, 0, cmd) then: (DONE, ‘done‘)
	 if (‘Active‘, 0, cmd) then: (CMD, cmd)
	 if (‘Error‘, 0, cmd) then: (DONE, ‘error‘)

ta	$\mathrm{ta}((\mathrm{phase}, \sigma, \mathrm{cmd})) := \sigma$

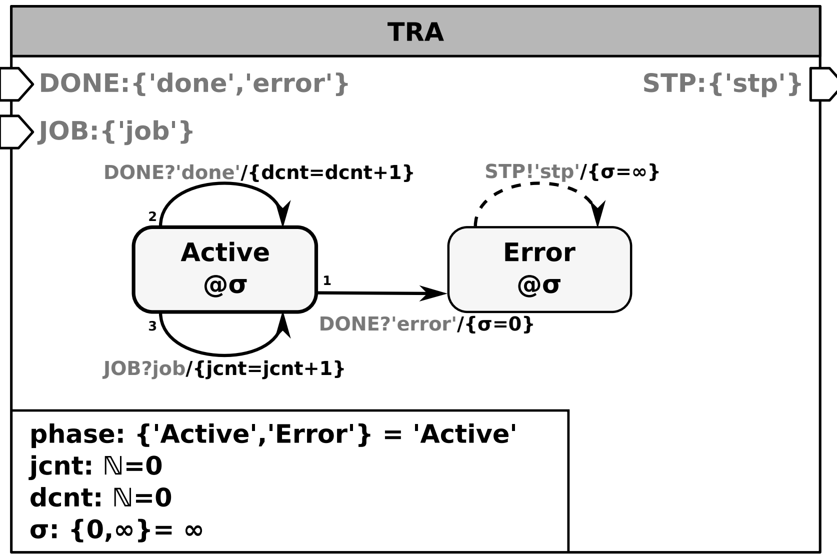


Abbildung B.4: Erweitertes DEVS-Diagramm TRA

Tabelle B.3: PDEVS-Mengennotation eines TRA

$\mathbf{TRA} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$	
X	$X = \{((\text{DONE}, \text{dval}), (\text{JOB}, \text{‘job‘}))\|\text{DONE}, \text{JOB} \in \text{IPorts}, \text{dval} \in \{\text{‘done‘}, \text{‘error‘}\}\}$
S	$S = \{(\text{phase}, \sigma, \text{jcnt}, \text{dcnt})\|\text{phase} \in \{\text{‘Active‘}, \text{‘Passive‘}\}, \sigma \in \mathbb{R}^+_{0,\infty}, \text{jcnt}, \text{dcnt} \in \mathbb{N}_0\}$
	$s_{\text{initial}} = (\text{‘Active‘}, \infty, 0, 0)$
Y	$Y = \{(\text{STP}, \text{‘stp‘})\|\text{STP} \in \text{OPorts}\}$
$\delta_{\mathbf{int}}$	$\delta_{\text{int}}((\text{phase}, \sigma, \text{jcnt}, \text{dcnt})) := (\text{‘Error‘}, \infty, \text{jcnt}, \text{dcnt})$
$\delta_{\mathbf{ext}}$	$\delta_{\text{ext}}((\text{phase}, \sigma, \text{jcnt}, \text{dcnt}), \text{e}, ((\text{DONE}, \text{dval}), (\text{JOB}, \text{‘job‘}))) :$
	if (DONE, ‘done‘) then: dcnt = dcnt + 1
	if (DONE, ‘error‘) then: phase = ‘Error‘, $\sigma = 0$
	if (JOB, ‘job‘) then: jcnt = jcnt + 1
$\delta_{\mathbf{con}}$	$\delta_{\text{con}}((\text{phase}, \sigma, \text{jcnt}, \text{dcnt}), \text{e}, ((\text{DONE}, \text{dval}), (\text{JOB}, \text{‘job‘}))) :=$
	$\delta_{\text{ext}}(\delta_{\text{int}}((\text{phase}, \sigma, \text{jcnt}, \text{dcnt})), \text{e}, ((\text{DONE}, \text{dval}), (\text{JOB}, \text{‘job‘})))$
λ	$\lambda((\text{phase}, \sigma, \text{jcnt}, \text{dcnt})) := (\text{STP}, \text{‘stp‘})$
ta	$\text{ta}((\text{phase}, \sigma, \text{jcnt}, \text{dcnt})) := \sigma$

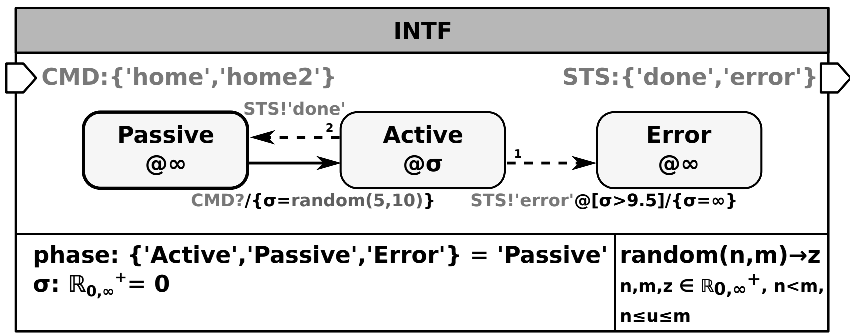


Abbildung B.5: Erweitertes DEVS-Diagramm INTF (Automatisierungsphase)

Tabelle B.4: PDEVS-Mengennotation eines INTF (Automatisierungspahse)

$\mathbf{INTF} = (\mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$	
$\mathbf{X}$	$\mathrm{X} = \{(\mathrm{CMD}, \mathrm{cmd}) \mid \mathrm{CMD} \in \mathrm{IPorts}, \mathrm{cmd} \in \{\text{‘home‘}, \text{‘home2‘}\}\}$
$\mathbf{S}$	$\mathrm{S} = \{(\mathrm{phase}, \sigma) \mid \mathrm{phase} \in \{\text{‘Active‘}, \text{‘Passive‘}, \text{‘Error‘}\}, \sigma \in \mathbb{R}^{+}_{0,\infty}\}$
	$\mathrm{s}_{\mathrm{initial}} = (\text{‘Passive‘}, \infty)$
$\mathbf{Y}$	$\mathrm{Y} = \{(\mathrm{STS}, \mathrm{sts}) \mid \mathrm{STS} \in \mathrm{OPorts}, \mathrm{sts} \in \{\text{‘done‘}, \text{‘error‘}\}\}$
$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}}((\mathrm{phase}, \sigma))$:
	if $(\sigma > 9.5)$ then: $(\text{‘Error‘}, \infty)$
	else $(\text{‘Passive‘}, \infty)$
$\delta_{\mathbf{ext}}$	$\delta_{\mathrm{ext}}((\mathrm{phase}, \sigma), \mathrm{e}, (\mathrm{CMD}, \mathrm{cmd})) := (\text{‘Active‘}, \mathrm{random}(5, 10)) \mid$
	$\mathrm{random}(\mathrm{n}, \mathrm{m}) \rightarrow \mathrm{z} \quad \mathrm{n}, \mathrm{m}, \mathrm{z} \in \mathbb{R}^{+}_{0,\infty}, \mathrm{n} < \mathrm{m}, \mathrm{n} \leq \mathrm{z} \leq \mathrm{m}$
$\delta_{\mathbf{con}}$	$\delta_{\mathrm{con}}((\mathrm{phase}, \sigma), \mathrm{e}, (\mathrm{CMD}, \mathrm{cmd})) :=$
	$\delta_{\mathrm{ext}}(\delta_{\mathrm{int}}((\mathrm{phase}, \sigma)), \mathrm{e}, (\mathrm{CMD}, \mathrm{cmd}))$
λ	$\lambda((\mathrm{phase}, \sigma))$:
	if $(\sigma > 9.5)$ then: $(\mathrm{STS}, \text{‘error‘})$
	else $(\mathrm{STS}, \text{‘done‘})$
$\mathbf{ta}$	$\mathrm{ta}((\mathrm{phase}, \sigma)) := \sigma$

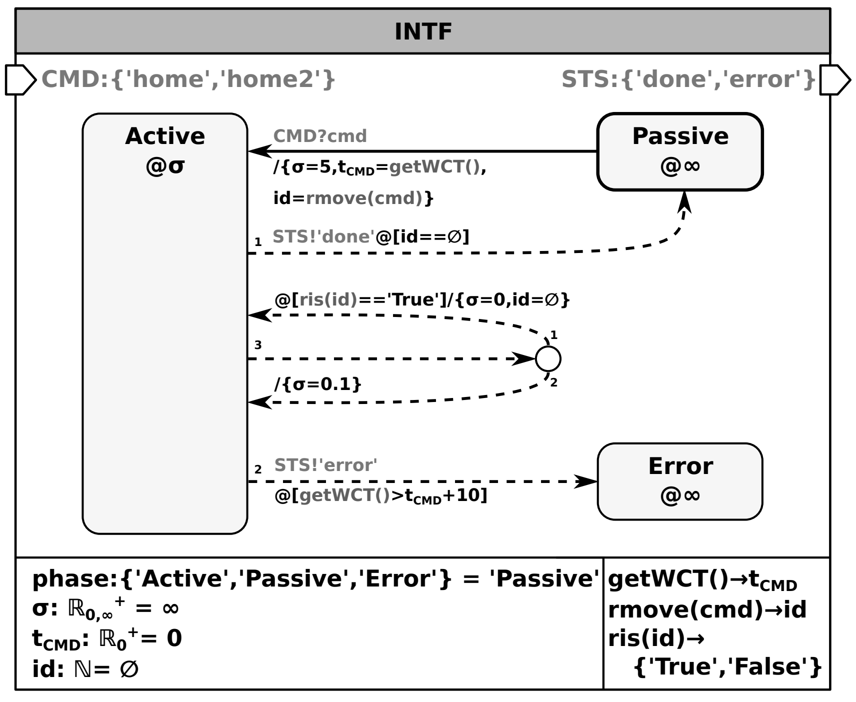


Abbildung B.6: Erweitertes DEVS-Diagramm INTF (operativer Betrieb)

Tabelle B.5: PDEVS-Mengennotation eines INTF (operativer Betrieb)

$\mathbf{INTF} = (\mathbf{X}, \mathbf{S_{RCP}}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{con}}, \lambda, \mathbf{ta})$	
X	$\mathrm{X} = \{(\mathrm{CMD}, \mathrm{cmd}) \mid \mathrm{CMD} \in \mathrm{IPorts}, \mathrm{cmd} \in \{\text{‘home‘}, \text{‘home2‘}\}\}$
$\mathbf{S_{RCP}}$	$\mathrm{S_{RCP}} = \mathrm{S} \cup \mathrm{A}$
	$\mathrm{S} = \{(\mathrm{phase}, \sigma, \mathrm{t_{CMD}}, \mathrm{id}) \mid \mathrm{phase} \in \{\text{‘Active‘}, \text{‘Passive‘}, \text{‘Error‘}\},$
	$\sigma \in \mathbb{R}^+_{0,\infty}, \mathrm{t_{CMD}} \in \mathbb{R}^+_0, \mathrm{id} \in \mathbb{N}_0\}$
	$\mathrm{s_{initial}} = (\text{‘Passive‘}, \infty, 0, \varnothing)$
	$\mathrm{A} = \{\mathrm{getWCT}() \rightarrow \mathbb{R}^+_{0,\infty}, \mathrm{rmove(cmd)} \rightarrow \mathrm{id}, \mathrm{ris(id)} \rightarrow \{\text{‘True‘}, \text{‘False‘}\}\}$
Y	$\mathrm{Y} = \{(\mathrm{STS}, \mathrm{sts}) \mid \mathrm{STS} \in \mathrm{OPorts}, \mathrm{sts} \in \{\text{‘done‘}, \text{‘error‘}\}\}$
$\delta_{\mathbf{int}}$	$\delta_{\mathrm{int}}((\mathrm{phase}, \sigma, \mathrm{t_{CMD}}, \mathrm{id})):$
	if $(\mathrm{id} == \varnothing)$ then: $\mathrm{phase} = \text{‘Passive‘}, \sigma = \infty$
	elseif $(\mathrm{getWCT}() > \mathrm{t_{CMD}} + 10)$ then: $\mathrm{phase} = \text{‘Error‘}, \sigma = \infty$
	elseif $(\mathrm{ris(id)} == \text{‘True‘})$ then: $\sigma = 0, \mathrm{id} = \varnothing$
	else: $\sigma = 0.1$
$\delta_{\mathbf{ext}}$	$\delta_{\mathrm{ext}}((\text{‘Passive‘}, \infty, \mathrm{t_{CMD}}, \varnothing), \mathrm{e}, (\mathrm{CMD}, \mathrm{cmd})) :=$
	$(\text{‘Active‘}, 5, \mathrm{getWCT}(), \mathrm{rmove(cmd)})$
$\delta_{\mathbf{con}}$	$\delta_{\mathrm{con}}((\mathrm{phase}, \sigma, \mathrm{t_{CMD}}, \mathrm{id}), \mathrm{e}, (\mathrm{CMD}, \mathrm{cmd})) :=$
	$\delta_{\mathrm{ext}}(\delta_{\mathrm{int}}((\mathrm{phase}, \sigma, \mathrm{t_{CMD}}, \mathrm{id})), \mathrm{e}, (\mathrm{CMD}, \mathrm{cmd}))$
λ	$\lambda((\mathrm{phase}, \sigma, \mathrm{t_{CMD}}, \mathrm{id})):$
	if $(\mathrm{id} == \varnothing)$ then: $(\mathrm{STS}, \text{‘done‘})$
	elseif $(\mathrm{getWCT}() > \mathrm{t_{CMD}} + 10)$ then: $(\mathrm{STS}, \text{‘error‘})$
	else nothing
ta	$\mathrm{ta}((\mathrm{phase}, \sigma, \mathrm{t_{CMD}}, \mathrm{id})) := \sigma$

C Lösungsansatz zu Interaktionsklasse 6

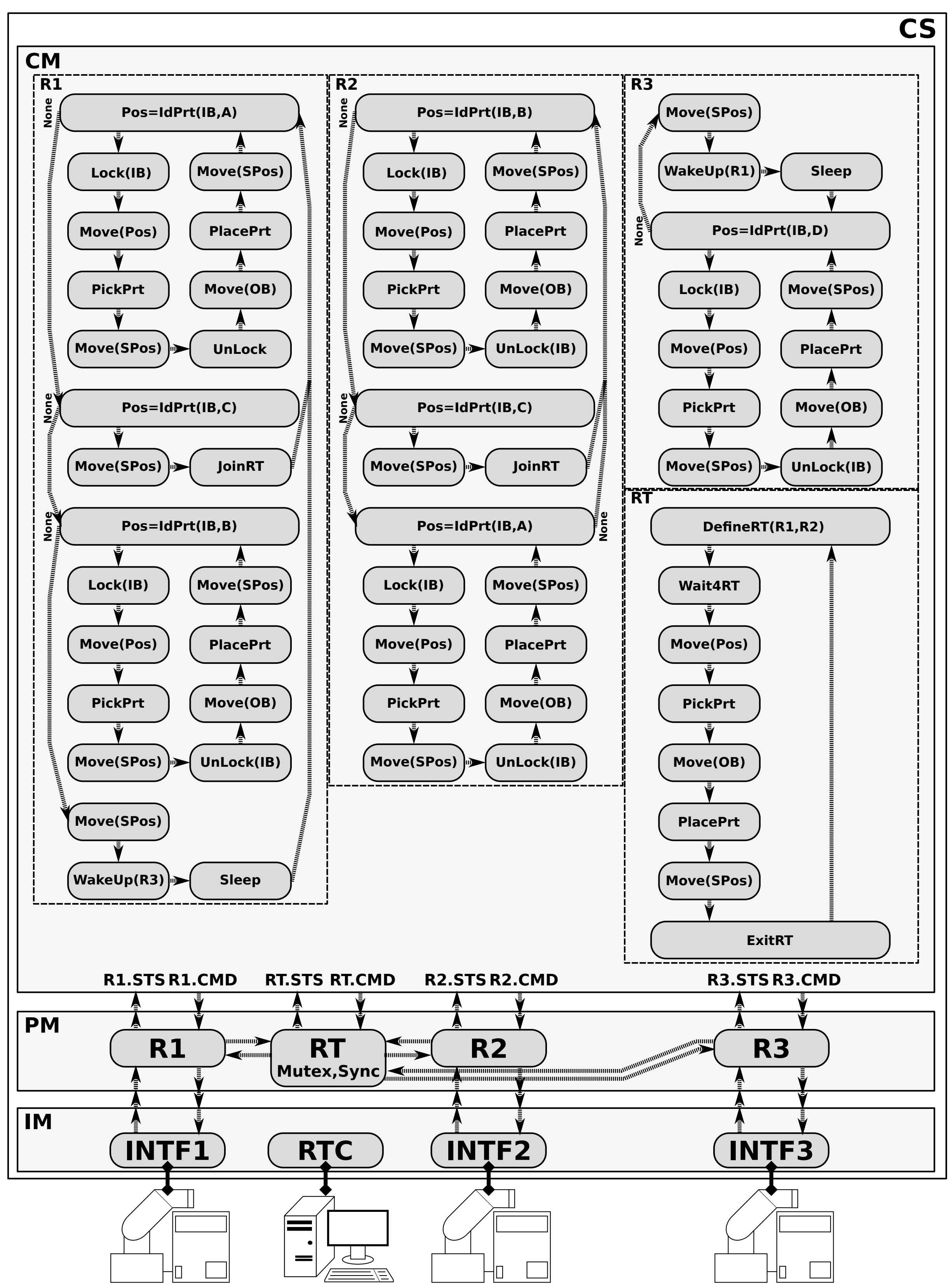
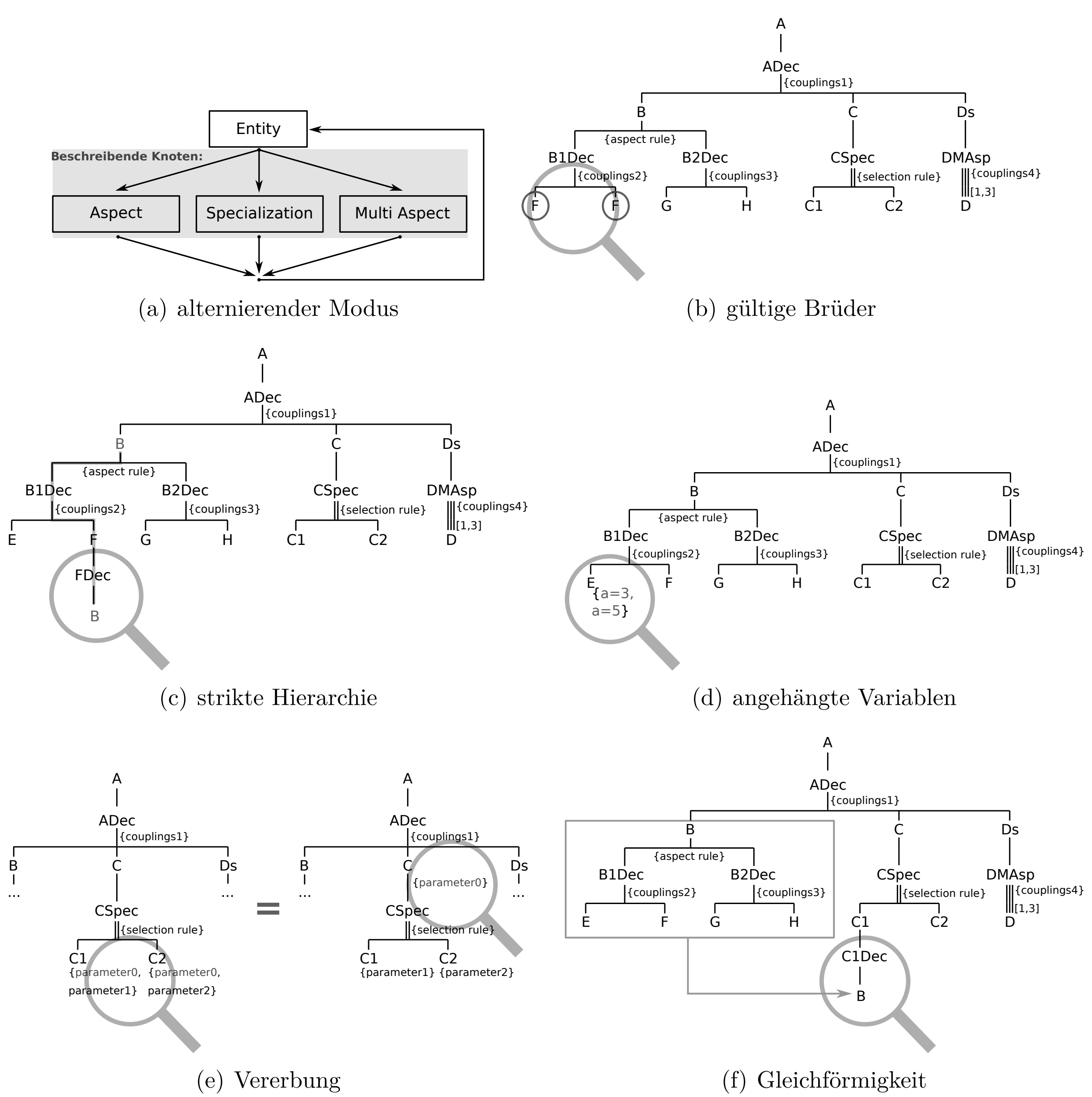


Abbildung C.7: Interaktionsklasse 6 ohne komponierte Aufgaben

D Axiome der System-Entity-Structure (SES)



(a) alternierender Modus

(b) gültige Brüder

(c) strikte Hierarchie

(d) angehängte Variablen

(e) Vererbung

(f) Gleichförmigkeit

Abbildung D.8: Beispiele zu den Axiomen der SES

E MATLAB/DEVS Quellcode zu Kapitel 6

Die Ausführung des nachfolgenden Quellcodes erfordert MATLAB und die Installation der *MATLAB/DEVS* Toolbox [28] und der *Robotic Control and Visualisation* Toolbox [20]. Tabelle E.6 listet die entwickelten PDEVS-Modelle zu Kapitel 6 auf. Die Bezeichner Class0 - Class4 beziehen sich auf die Interaktionsklassen aus Kapitel 4. Die Implementierung erfolgt analog der Beispiele in Kapitel 6.

Tabelle E.6: Liste der entwickelten PDEVS-Modelle zu Kapitel 6

Modell	**Beschreibung**
GTask	CM-Ebene des SBC, atomic-PDEVS einer allgemeinen Aufgabe
IdPrt	CM-Ebene des SBC, atomic-PDEVS der Aufgabe Identify Part
NoOp	CM-Ebene des SBC, atomic-PDEVS der Aufgabe No Operation
R	PM-Ebene des SBC, atomic-PDEVS Roboter
RT	PM-Ebene des SBC, atomic-PDEVS Roboter-Team
INTF	IM-Ebene des SBC, atomic-PDEVS (Roboter-) Interface
RTC	IM-Ebene des SBC, atomic-PDEVS Echtzeituhr
Class0	Coupled-PDEVS, Beispiel zur Interaktionsklasse 0 (SRS)
Class1u2	Coupled-PDEVS, Beispiel zur Interaktionsklasse 1 und 2 (MRS)
Class3	Coupled-PDEVS, Beispiel zur Interaktionsklasse 3 (MRS)
Class4	Coupled-PDEVS, Beispiel zur Interaktionsklasse 4 (MRS)

Tabelle E.7: Liste der entwickelten PDEVS-Modelle zu Kapitel 6

Modell	**Beschreibung**	**Listing**
GTask	CM-Ebene des SBC, atomic-PDEVS einer allgemeinen Aufgabe	G.1
IdPrt	CM-Ebene des SBC, atomic-PDEVS der Aufgabe Identify Part	G.2
NoOp	CM-Ebene des SBC, atomic-PDEVS der Aufgabe No Operation	G.3
R	PM-Ebene des SBC, atomic-PDEVS Roboter	G.4
RT	PM-Ebene des SBC, atomic-PDEVS Roboter-Team	G.5
INTF	IM-Ebene des SBC, atomic-PDEVS (Roboter-) Interface	G.6
RTC	IM-Ebene des SBC, atomic-PDEVS Echtzeituhr	G.7
Class0	Coupled-PDEVS, Beispiel zur Interaktionsklasse 0 (SRS)	G.8
Class1u2	Coupled-PDEVS, Beispiel zur Interaktionsklasse 1 und 2 (MRS)	G.9
Class3	Coupled-PDEVS, Beispiel zur Interaktionsklasse 3 (MRS)	G.10
Class4	Coupled-PDEVS, Beispiel zur Interaktionsklasse 4 (MRS)	G.11

Der Quellcode ist auf der beigefügten CD enthalten.

Listing G.1: Atomic-PDEVS-Modell der Aufgabe GTask

```
classdef GTask < atomic
 methods
   function obj = GTask(name,ini_struct)
     x = {'BEG' 'STS' };
     y = {'NXT' 'CMD' };
     s = {'phase' 'sigma'};
     elapsed = 0;
     sysparams = ini_struct;
     obj = obj@atomic(name,x,y,s,elapsed,sysparams);

     obj.s.phase = 'Passive';
     obj.s.sigma = inf;
   end

   function ta = tafun(obj)
     ta = obj.s.sigma;
   end
```

```

    function deltaconffun(obj,gt)
      deltaextfun(obj,gt);
      deltaintfun(obj);
    end

    function deltaextfun(obj,gt)
      switch obj.s.phase
        case ’Passive’
          if strcmp(obj.x.BEG,’next’)
            obj.s.phase = ’Active’;
            obj.s.sigma = 0;
          end
        case ’Active’
          if strcmp(obj.x.STS,’done’)
            obj.s.phase = ’Passive’;
            obj.s.sigma = 0;
          end
        otherwise
          error(obj.s.phase)
      end
    end

    function deltaintfun(obj)
      obj.s.sigma = inf;
    end

    function lambdafun(obj)
      switch obj.s.phase
        case ’Passive’
          obj.y.NXT = {’next’};
        case ’Active’
          sysP = obj.sysparams;
          obj.y.CMD = {sysP.tid, sysP.p1 sysP.p2};
        otherwise
          error(obj.s.phase)
      end
    end
  end
end
```

Listing G.2: Atomic-PDEVS-Modell der Aufgabe IdPrt

```
classdef IdPrt < atomic
  methods
    function obj = IdPrt(name,ini_struct)
      x = {’BEG’ ’STS’ };
      y = {’NXT’ ’CMD’ ’NONE’};
      s = {’phase’ ’sigma’ ’res’ ’p1’ ’p2’ };
      elapsed = 0;
      sysparams = ini_struct;
      obj = obj@atomic(name,x,y,s,elapsed,sysparams);

      obj.s.phase = ’Passive’;
      obj.s.sigma = inf;
    end

    function ta = tafun(obj)
```

```
      ta = obj.s.sigma;
    end

    function deltaconffun(obj,gt)
      deltaextfun(obj,gt);
      deltaintfun(obj);
    end

    function deltaextfun(obj,gt)
      switch obj.s.phase
        case 'Passive'
          if strcmp(obj.x.BEG,'next')
            obj.s.phase = 'Active';
            obj.s.sigma = 0;
          end
        case 'Active'
          if strcmp(obj.x.STS,'done')
            obj.s.phase = 'Passive';
            obj.s.sigma = 0;
            obj.s.res = obj.x.STS{1};
          elseif strcmp(obj.x.STS,'none')
            obj.s.phase = 'Passive';
            obj.s.sigma = 0;
            obj.s.res = obj.x.STS{:};
          else
            error(obj.x.STS)
          end
        otherwise
          error(obj.s.phase)
      end
    end

    function deltaintfun(obj)
      obj.s.sigma = inf;
    end

    function lambdafun(obj)
      switch obj.s.phase
        case 'Passive'
          if strcmp(obj.s.res,'done')
            obj.y.NXT = {'next'};
          elseif strcmp(obj.s.res,'none')
            obj.y.NONE = {'next'};
          end
        case 'Active'
          sysP = obj.sysparams;
          obj.y.CMD = {sysP.tid,sysP.p1, sysP.p2};
        otherwise
          error(obj.s.phase)
      end
    end
  end
end
```

Listing G.3: Atomic-PDEVS-Modell der Aufgabe NoOp

```
classdef NoOp < atomic
  methods
```

```
    function obj = NoOp(name)
      x = {};
      y = {'NXT' };
      s = {'phase' };
      elapsed = 0;
      obj = obj@atomic(name,x,y,s,elapsed,[]);

      obj.s.phase = 'Active';
    end

    function ta = tafun(obj)
      if strcmp(obj.s.phase,'Passive')
        ta = inf;
      elseif strcmp(obj.s.phase,'Active')
        ta = 0;
      end
    end

    function deltaconffun(obj,gt)
    end

    function deltaextfun(obj,gt)
    end

    function deltaintfun(obj)
      obj.s.phase = 'Passive';
    end

    function lambdafun(obj)
      if strcmp(obj.s.phase,'Active')
        obj.y.NXT = {'next'};
      end
    end
  end
end
```

Listing G.4: Atomic-PDEVS-Modell der Komponente R

```
classdef R < atomic
  methods
    function obj = R(name)
      x = {'CM_CMD' 'RT_IN' 'IM_STS' };
      y = {'CM_STS' 'RT_OUT' 'IM_CMD' };
      s = {'phase' 'sigma' 'tid' 'p1' 'p2' 'rtcmd' 'sts' ...
        'position' 'cmd' };
      elapsed = 0;
      sysparams = struct('rid', name);
      obj = obj@atomic(name,x,y,s,elapsed,sysparams);

      obj.s.phase    = 'Passive';
      obj.s.sigma    = inf;
      obj.s.tid      = [];
      obj.s.p1       = '';
      obj.s.p2       = '';
      obj.s.rtcmd    = [];
      obj.s.sts      = '';
      obj.s.position = [];
      obj.s.cmd      = [];
```

```
    end
22
    function ta = tafun(obj)
24    if strcmp(obj.s.phase,'Error')
        ta = inf;
26    else
        ta = obj.s.sigma;
28    end
    end
30
    function deltaconffun(obj,gt)
32    deltaextfun(obj,gt);
      deltaintfun(obj);
34  end

36  function deltaextfun(obj,gt)
      switch obj.s.phase
38      case 'Passive'
          if ~isempty(obj.x.CM_CMD)
40          [obj.s.tid, obj.s.p1, obj.s.p2] = obj.x.CM_CMD{1:3};
            obj.s.sigma = 0;
42          switch obj.s.tid
              case  1 %PickPrt(p1,p2)
44              obj.s.phase = 'Active';
                obj.s.cmd = 'CloseG';
46            case  2 %PlacePrt
                obj.s.phase = 'Active';
48              obj.s.cmd = 'OpenG';
              case  3 %IdPrt
50              p1 = obj.s.p1;
                p2 = obj.s.p2;
52              [obj.s.position, obj.s.sts] = LookUpTbl1(p1,p2);
              case  4 %Move(p1)
54              obj.s.phase = 'Active';
                if strcmp(obj.s.p1,'Pos')
56                obj.s.cmd = obj.s.position;
                else
58                p1 = obj.s.p1;
                  obj.s.cmd = LookUpTbl2(p1);
60              end
              case -1 %Lock(p1)
62              obj.s.phase = 'PassiveRT';
              case -2 %UnLock(p1)
64              obj.s.phase = 'PassiveRT';
              case -3 %JoinRT(p1)
66              obj.s.phase = 'PassiveRT';
              case -7 %Sleep
68              obj.s.phase = 'PassiveRT';
                obj.s.sigma = inf;
70            case -8 %WakeUp(p1)
                obj.s.phase = 'PassiveRT';
72          end
          end
74      case 'Active'
          if ~isempty(obj.x.IM_STS)
76          if strcmp(obj.x.IM_STS,'done')
              obj.s.phase = 'Passive';
78            obj.s.sts = 'done';
```

```
              obj.s.sigma = 0;
            elseif strcmp(obj.x.IM_STS,'error')
              obj.s.phase = 'Error';
            end
          end

        case 'PassiveRT'
          if ~isempty(obj.x.RT_IN)
            if strcmp(obj.x.RT_IN,'done')
              obj.s.phase = 'Passive';
              obj.s.sts = 'done';
              obj.s.sigma = 0;
            else
              obj.s.rtcmd = obj.x.RT_IN{1};
              obj.s.sigma = 0;
            end
          end

        case 'ActiveRT'
          if strcmp(obj.x.IM_CMD,'error')
            obj.s.phase = 'Error';
          elseif strcmp(obj.x.IM_CMD,'done')
            obj.s.phase = 'PassiveRT';
            obj.s.p1 = 'done';
            obj.s.sigma = 0;
          end
        otherwise
          error('PHASE NOT ALLOWED')
      end
    end

    function deltaintfun(obj)
      obj.s.sigma = inf;
    end

    function lambdafun(obj)
      switch obj.s.phase
        case 'Passive'
          obj.y.CM_STS = {obj.s.sts};
        case 'Active'
          obj.y.IM_CMD = {obj.s.cmd};
        case 'PassiveRT'
          obj.y.RT_OUT = {obj.s.tid, obj.sysparams.rid, obj.s.p1};
        case 'ActiveRT'
          obj.y.IM_CMD = {obj.s.rtcmd};
        otherwise
          error(obj.s.phase)
      end
    end
  end
end
```

Listing G.5: Atomic-PDEVS-Modell der Komponente RT

```
classdef RT < atomic
  methods
    function obj = RT(name)
      x = {'CM_CMD' 'R1_IN'  'R2_IN'  'R3_IN' };
```

```
    y = {'CM_STS' 'R1_OUT' 'R2_OUT' 'R3_OUT'};
6   s = {'phase' 'sigma' ...
      'rid1' 'rid2' 'rid3' 'tid0' 'tid1' 'tid2' 'tid3' ...
8     'cp1' 'cp2' 'p1' 'p2' 'p3' 'position'...
      'cmd0' 'cmd1' 'cmd2' 'cmd3' ...
10    'RTnList' 'RTwList' 'RcsList'};
    elapsed = 0;
12  sysparams = struct('rid', name);
    obj = obj@atomic(name,x,y,s,elapsed,sysparams);
14
    obj.s.phase     = 'Active';
16  obj.s.sigma     = inf;
    obj.s.rid1      = [];
18  obj.s.rid2      = [];
    obj.s.rid3      = [];
20  obj.s.tid0      = 0;
    obj.s.tid1      = 0;
22  obj.s.tid2      = 0;
    obj.s.tid3      = 0;
24  obj.s.cp1       = [];
    obj.s.cp2       = [];
26  obj.s.p1        = [];
    obj.s.p2        = [];
28  obj.s.p3        = [];
    obj.s.position  = [];
30  obj.s.cmd0      = [];
    obj.s.cmd1      = [];
32  obj.s.cmd2      = [];
    obj.s.cmd3      = [];
34  obj.s.RTnList   = {};
    obj.s.RTwList   = {};
36  obj.s.RcsList   = struct();
  end
38
  function ta = tafun(obj)
40  ta = obj.s.sigma;
  end
42
  function deltaconffun(obj,gt)
44  deltaextfun(obj,gt);
  end
46
  function deltaextfun(obj,gt)
48  obj.s.sigma = 0;
    if ~isempty(obj.x.CM_CMD)
50    [obj.s.tid0, obj.s.cp1, obj.s.cp2] = obj.x.CM_CMD{1:3};
    end
52  if ~isempty(obj.x.R1_IN)
      [obj.s.tid1, obj.s.rid1, obj.s.p1] = obj.x.R1_IN{1:3};
54  end
    if ~isempty(obj.x.R2_IN)
56    [obj.s.tid2, obj.s.rid2, obj.s.p2] = obj.x.R2_IN{1:3};
    end
58  if ~isempty(obj.x.R3_IN)
      [obj.s.tid3, obj.s.rid3, obj.s.p3] = obj.x.R3_IN{1:3};
60  end

62  %R1.UnLock(p1)
```

```
      if obj.s.tid1 == -2
64      if obj.s.RcsList.(obj.s.p1)== obj.s.rid1
          obj.s.RcsList.(obj.s.p1)= [];
66      end
        obj.s.cmd1 = 'done';
68      obj.s.tid1 = 0;
      end
70
      %R2.UnLock(p2)
72    if obj.s.tid2 == -2
        if obj.s.RcsList.(obj.s.p2)== obj.s.rid2
74        obj.s.RcsList.(obj.s.p2)= [];
        end
76      obj.s.cmd2 = 'done';
        obj.s.tid2 = 0;
78    end

80    %R2.UnLock(p3)
      if obj.s.tid3 == -2
82      if obj.s.RcsList.(obj.s.p3)== obj.s.rid3
          obj.s.RcsList.(obj.s.p3)= [];
84      end
        obj.s.cmd3 = 'done';
86      obj.s.tid3 = 0;
      end
88
      %R1.Lock(p1)
90    if obj.s.tid1 == -1
        if ~isfield(obj.s.RcsList,obj.s.p1) || isempty(obj.s.RcsList.(obj.s.p1))
92        obj.s.RcsList.(obj.s.p1) = obj.s.rid1;
          obj.s.cmd1 = 'done';
94        obj.s.tid1 = 0;
        end
96    end

98    %R2.Lock(p2)
      if obj.s.tid2 == -1
100     if ~isfield(obj.s.RcsList,obj.s.p2) || isempty(obj.s.RcsList.(obj.s.p2))
          obj.s.RcsList.(obj.s.p2) = obj.s.rid2;
102       obj.s.cmd2 = 'done';
          obj.s.tid2 = 0;
104     end
      end
106
      %R3.Lock(p3)
108   if obj.s.tid3 == -1
        if ~isfield(obj.s.RcsList,obj.s.p3) || isempty(obj.s.RcsList.(obj.s.p3))
110       obj.s.RcsList.(obj.s.p3) = obj.s.rid3;
          obj.s.cmd3 = 'done';
112       obj.s.tid3 = 0;
        end
114   end

116   %R1.JoinRT(rid1)
      if obj.s.tid1 == -3
118     obj.s.RTnList{1} = obj.s.rid1;
        obj.s.tid1 = 0;
120   end
```

```
122       %R2.JoinRT(rid2)
          if obj.s.tid2 == -3
124         obj.s.RTnList{2} = obj.s.rid2;
            obj.s.tid2 = 0;
126       end

128       %R3.JoinRT(rid3)
          if obj.s.tid3 == -3
130         obj.s.RTnList{3} = obj.s.rid3;
            obj.s.tid3 = 0;
132       end

134       %R1.WakeUp(p1)
          if obj.s.tid1 == -8
136         if strcmp(obj.s.p1,'R2')
              obj.s.cmd2 = 'done';
138         else
              obj.s.cmd3 = 'done';
140         end
            obj.s.cmd1 = 'done';
142         obj.s.tid1 = 0;
          end
144
          %R2.WakeUp(p2)
146       if obj.s.tid2 == -8
            if strcmp(obj.s.p2,'R1')
148           obj.s.cmd1 = 'done';
            else
150           obj.s.cmd3 = 'done';
            end
152         obj.s.cmd2 = 'done';
            obj.s.tid2 = 0;
154       end

156       %R3.WakeUp(p3)
          if obj.s.tid3 == -8
158         if strcmp(obj.s.p3,'R1')
              obj.s.cmd1 = 'done';
160         else
              obj.s.cmd2 = 'done';
162         end
            obj.s.cmd3 = 'done';
164         obj.s.tid3 = 0;
          end
166
          %DefineRT(cp1,cp2)
168       if obj.s.tid0 == -4
            obj.s.RTwList{1} = obj.s.cp1;
170         obj.s.RTwList{2} = obj.s.cp2;
            obj.s.cmd0 = 'done';
172         obj.s.tid0 = 0;
          end
174
          %Wait4RT()
176       if obj.s.tid0 == -5
            try
178           if(ismember(obj.s.RTnList,obj.s.RTwList))
```

```
          obj.s.cmd0 = 'done';
180       obj.s.tid0 = 0;
        end
182     catch
      end
184   end

186   %ExitRT()
      if obj.s.tid0 == -6
188     obj.s.cmd1 = 'done';
        obj.s.cmd2 = 'done';
190     obj.s.cmd0 = 'done';
        obj.s.tid0 = 0;
192   end

194   %Move(cp1)
      if obj.s.tid0 == 4
196     [obj.s.cmd1, obj.s.cmd2, obj.s.cmd3] = LookUpTbl(obj.s.cp1);
        obj.s.tid0 = 0;
198   end

200   %PickPrt()
      if obj.s.tid0 == 1
202     obj.s.cmd1 = 'CloseG';
        obj.s.cmd2 = 'CloseG';
204     obj.s.tid0 = 0;
      end
206
      %PickPrt()
208   if obj.s.tid0 == 2
        obj.s.cmd1 = 'OpenG';
210     obj.s.cmd2 = 'OpenG';
        obj.s.tid0 = 0;
212   end

214   %Check Team
      if obj.s.tid0 == 0
216     if strcmp(obj.s.p1,'done') && strcmp(obj.s.p2,'done')
          obj.s.p1 = '';
218       obj.s.p2 = '';
          obj.s.cmd0 = 'done';
220     end
      end
222 end

224 function deltaintfun(obj)
      obj.s.sigma = inf;
226   obj.s.cmd0  = [];
      obj.s.cmd1  = [];
228   obj.s.cmd2  = [];
      obj.s.cmd3  = [];
230 end

232 function lambdafun(obj)
      switch obj.s.phase
234     case 'Active'
          if ~isempty(obj.s.cmd0)
236         obj.y.CM_STS = {obj.s.cmd0};
```

```
          end
          if ~isempty(obj.s.cmd1)
            obj.y.R1_OUT = {obj.s.cmd1};
          end
          if ~isempty(obj.s.cmd2)
            obj.y.R2_OUT = {obj.s.cmd2};
          end
          if ~isempty(obj.s.cmd3)
            obj.y.R3_OUT = {obj.s.cmd3};
          end
        otherwise
          error(obj.s.phase)
      end
      obj.y
    end
  end
end
```

Listing G.6: Atomic-PDEVS-Modell der Komponente INTF

```
classdef INTF < atomic
  methods
    function obj = INTF(name,ini_struct)
      x = {'CMD' };
      y = {'STS' };
      s = {'phase' 'sigma' 'tCMD' 'id' };
      elapsed = 0;
      sysparams = ini_struct;
      obj = obj@atomic(name,x,y,s,elapsed,sysparams);

      obj.s.phase    = 'Passive';
      obj.s.sigma    = inf;
      obj.s.tCMD     = 0;
      obj.s.id       = [];
    end

    function ta = tafun(obj)
      switch obj.s.phase
        case {'Passive' 'Error'}
          ta = inf;
        otherwise
          ta = obj.s.sigma;
      end
    end

    function deltaconffun(obj,gt)
      deltaextfun(obj,gt);
      deltaintfun(obj);
    end

    function deltaextfun(obj,gt)
      if strcmp(obj.s.phase,'Passive')
        cmd = obj.x.CMD{1};
        if ischar(cmd)
          if strcmp(cmd,'OpenG')
            rset(obj.sysparams.rid, 'signal', [9, -10])
            obj.s.sigma = 1;
          elseif strcmp(cmd,'CloseG')
```

```
            rset(obj.sysparams.rid, 'signal', [-9, 10])
40          obj.s.sigma = 1;
          end
42        obj.s.id    = [];
          obj.s.sigma = 0;
44      else
          obj.s.id    = rmove(obj.sysparams.rid, cmd);
46        obj.s.tCMD  = toc();
          obj.s.sigma = 5;
48      end
        obj.s.phase = 'Active';
50    end
    end
52
    function deltaintfun(obj)
54    if strcmp(obj.s.phase,'Active')
        if isempty(obj.s.id)
56        obj.s.phase = 'Passive';
        elseif toc > (obj.s.tCMD + 15)
58        obj.s.phase = 'Error';
        elseif ris(obj.s.id)
60        obj.s.id    = [];
          obj.s.sigma = 0;
62      else
          obj.s.sigma = 0.1;
64      end
      end
66  end

68  function lambdafun(obj)
      if strcmp(obj.s.phase,'Active')
70      if isempty(obj.s.id)
          obj.y.STS = {'done'};
72      elseif toc > (obj.s.tCMD + 15)
          obj.y.STS = {'error'};
74      end
      end
76  end
  end
78 end
```

Listing G.7: Atomic-PDEVS-Modell der Komponente RTC

```
classdef RTC < atomic
2  methods
    function obj = RTC(name)
4     if nargin == 0
        name = mfilename;
6     end
      x = {};
8     y = {};
      s = {'phase' 'sigma' 'tLast' 'tWCT' };
10    obj = obj@atomic(name,x,y,s,0,[]);

12    obj.s.phase = 'Active';
      obj.s.sigma = 0;
14    obj.s.tLast = 0;
      obj.s.tWCT  = 0;
```

```
16     end
       function ta = tafun(obj)
18       ta = obj.s.sigma;
       end
20     function deltaconffun(obj,~)
       end
22     function deltaextfun(obj,~)
       end
24     function deltaintfun(obj)
         if ~obj.s.tLast
26         obj.s.tLast=toc;
         end
28
         obj.s.tWCT = toc;
30       if (obj.s.tWCT - obj.s.tLast) <= 0.1
           obj.s.sigma = 0;
32       else
           obj.s.sigma = 0.1;
34         obj.s.tLast = toc;
         end
36     end
       function lambdafun(~)
38     end
     end
40 end
```

Listing G.8: Coupled-PDEVS RM zur Interaktionsklasse 0

```
   RM = coupled('RM');
 2 RM.addcomponents({...
     ... %CM
 4   NoOp( 'NoOp'),...
     ... %CM (R1)
 6   IdPrt('R1_IdPrt_1',    struct('tid', 3, 'p1','IB',  'p2','A')),...
     GTask('R1_Move_1',     struct('tid', 4, 'p1','Pos', 'p2',[])),...
 8   GTask('R1_PickPrt_1',  struct('tid', 1, 'p1',[],    'p2',[])),...
     GTask('R1_Move_2',     struct('tid', 4, 'p1','OB',  'p2',[])),...
10   GTask('R1_PlacePrt_1', struct('tid', 2, 'p1',[],    'p2',[])),...
     ... %PM
12       R('R1'),...
     ... %IM
14   INTF( 'INTF1',         struct('rid',1)),...
     RTC(  'RTC')
16   });

18
   RM.set_Zid({
20   ... %CM
     'NoOp'          'NXT'     'R1_IdPrt_1'    'BEG';...
22   ... %CM (R1)
     'R1_IdPrt_1'    'NXT'     'R1_Move_1'     'BEG';...
24   'R1_Move_1'     'NXT'     'R1_PickPrt_1'  'BEG';...
     'R1_PickPrt_1'  'NXT'     'R1_Move_2'     'BEG';...
26   'R1_Move_2'     'NXT'     'R1_PlacePrt_1' 'BEG';...
     'R1_PlacePrt_1' 'NXT'     'R1_IdPrt_1'    'BEG';...
28   'R1_IdPrt_1'    'CMD'     'R1'            'CM_CMD';...
     'R1_Move_1'     'CMD'     'R1'            'CM_CMD';...
30   'R1_PickPrt_1'  'CMD'     'R1'            'CM_CMD';...
```

```
    'R1_Move_2'      'CMD'    'R1'             'CM_CMD';...
32  'R1_PlacePrt_1' 'CMD'    'R1'             'CM_CMD';...
    ... %PM (R1)
34  'R1'           'CM_STS'  'R1_IdPrt_1'    'STS';...
    'R1'           'CM_STS'  'R1_Move_1'     'STS';...
36  'R1'           'CM_STS'  'R1_PickPrt_1'  'STS';...
    'R1'           'CM_STS'  'R1_Move_2'     'STS';...
38  'R1'           'CM_STS'  'R1_PlacePrt_1' 'STS';...
    ... %IM (R1)
40  'R1'           'IM_CMD'  'INTF1'         'CMD';...
    'INTF1'        'STS'     'R1'            'IM_STS'
42  })
```

Listing G.9: Coupled-PDEVS RM zur Interaktionsklasse 1 und 2

```
   RM = coupled('RM');
 2 RM.addcomponents({...
    ... %CM
 4  NoOp( 'NoOp'),...
    ... %CM (R1)
 6  IdPrt('R1_IdPrt_1',    struct('tid', 3, 'p1','IB',  'p2','A')),...
    GTask('R1_Move_1',     struct('tid', 4, 'p1','Pos', 'p2',[])),...
 8  GTask('R1_PickPrt_1',  struct('tid', 1, 'p1',[],    'p2',[])),...
    GTask('R1_Move_2',     struct('tid', 4, 'p1','OB',  'p2',[])),...
10  GTask('R1_PlacePrt_1', struct('tid', 2, 'p1',[],    'p2',[])),...
    ... %CM (R2)
12  IdPrt('R2_IdPrt_1',    struct('tid', 3, 'p1','IB',  'p2','B')),...
    GTask('R2_Move_1',     struct('tid', 4, 'p1','Pos', 'p2',[])),...
14  GTask('R2_PickPrt_1',  struct('tid', 1, 'p1',[],    'p2',[])),....
    GTask('R2_Move_2',     struct('tid', 4, 'p1','OB',  'p2',[])),...
16  GTask('R2_PlacePrt_1', struct('tid', 2, 'p1',[],    'p2',[])),...
    ... %PM
18  R(    'R1'),...
    R(    'R2'),...
20  ... %IM
    INTF( 'INTF1',        struct('rid',1)),...
22  INTF( 'INTF2',        struct('rid',2)),...
    RTC(  'RTC')
24  });

26 RM.set_Zid({
    ... %CM
28  'NoOp'           'NXT'    'R1_IdPrt_1'    'BEG';...
    'NoOp'           'NXT'    'R2_IdPrt_1'    'BEG';...
30  ... %CM (R1)
    'R1_IdPrt_1'     'NXT'    'R1_Move_1'     'BEG';...
32  'R1_Move_1'      'NXT'    'R1_PickPrt_1'  'BEG';...
    'R1_PickPrt_1'   'NXT'    'R1_Move_2'     'BEG';...
34  'R1_Move_2'      'NXT'    'R1_PlacePrt_1' 'BEG';...
    'R1_PlacePrt_1'  'NXT'    'R1_IdPrt_1'    'BEG';...
36  'R1_IdPrt_1'     'CMD'    'R1'            'CM_CMD';...
    'R1_Move_1'      'CMD'    'R1'            'CM_CMD';...
38  'R1_PickPrt_1'   'CMD'    'R1'            'CM_CMD';...
    'R1_Move_2'      'CMD'    'R1'            'CM_CMD';...
40  'R1_PlacePrt_1'  'CMD'    'R1'            'CM_CMD';...
    ... %CM (R2)
42  'R2_IdPrt_1'     'NXT'    'R2_Move_1'     'BEG';...
    'R2_Move_1'      'NXT'    'R2_PickPrt_1'  'BEG';...
```

```
44  'R2_PickPrt_1'  'NXT'    'R2_Move_2'     'BEG';...
    'R2_Move_2'     'NXT'    'R2_PlacePrt_1' 'BEG';...
46  'R2_PlacePrt_1' 'NXT'    'R2_IdPrt_1'    'BEG';...
    'R2_IdPrt_1'    'CMD'    'R2'            'CM_CMD';...
48  'R2_Move_1'     'CMD'    'R2'            'CM_CMD';...
    'R2_PickPrt_1'  'CMD'    'R2'            'CM_CMD';...
50  'R2_Move_2'     'CMD'    'R2'            'CM_CMD';...
    'R2_PlacePrt_1' 'CMD'    'R2'            'CM_CMD';...
52  ... %PM (R1)
    'R1'            'CM_STS' 'R1_IdPrt_1'    'STS';...
54  'R1'            'CM_STS' 'R1_Move_1'     'STS';...
    'R1'            'CM_STS' 'R1_PickPrt_1'  'STS';...
56  'R1'            'CM_STS' 'R1_Move_2'     'STS';...
    'R1'            'CM_STS' 'R1_PlacePrt_1' 'STS';...
58  ... %PM (R2)
    'R2'            'CM_STS' 'R2_IdPrt_1'    'STS';...
60  'R2'            'CM_STS' 'R2_Move_1'     'STS';...
    'R2'            'CM_STS' 'R2_PickPrt_1'  'STS';...
62  'R2'            'CM_STS' 'R2_Move_2'     'STS';...
    'R2'            'CM_STS' 'R2_PlacePrt_1' 'STS';...
64  ... %IM (R1)
    'R1'            'IM_CMD' 'INTF1'         'CMD';...
66  'INTF1'         'STS'    'R1'            'IM_STS';...
    ... %IM (R2)
68  'R2'            'IM_CMD' 'INTF2'         'CMD';...
    'INTF2'         'STS'    'R2'            'IM_STS'
70  })
```

Listing G.10: Coupled-PDEVS RM zur Interaktionsklasse 3

```
   RM = coupled('RM');
 2 RM.addcomponents({...
    ... %CM
 4  NoOp( 'NoOp'),...
    ... %CM (R1)
 6  IdPrt('R1_IdPrt_1',    struct('tid', 3, 'p1','IB',   'p2','A')),...
    GTask('R1_Lock_1',     struct('tid',-1, 'p1','IB',   'p2',[])),...
 8  GTask('R1_Move_1',     struct('tid', 4, 'p1','Pos',  'p2',[])),...
    GTask('R1_PickPrt_1',  struct('tid', 1, 'p1',[],     'p2',[])),...
10  GTask('R1_Move_2',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
    GTask('R1_UnLock_1',   struct('tid',-2, 'p1','IB',   'p2',[])),...
12  GTask('R1_Move_3',     struct('tid', 4, 'p1','OB',   'p2',[])),...
    GTask('R1_PlacePrt_1', struct('tid', 2, 'p1',[],     'p2',[])),...
14  GTask('R1_Move_4',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
    ... %CM (R2)
16  IdPrt('R2_IdPrt_1',    struct('tid', 3, 'p1','IB',   'p2','B')),...
    GTask('R2_Lock_1',     struct('tid',-1, 'p1','IB',   'p2',[])),...
18  GTask('R2_Move_1',     struct('tid', 4, 'p1','Pos',  'p2',[])),...
    GTask('R2_PickPrt_1',  struct('tid', 1, 'p1',[],     'p2',[])),...
20  GTask('R2_Move_2',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
    GTask('R2_UnLock_1',   struct('tid',-2, 'p1','IB',   'p2',[])),...
22  GTask('R2_Move_3',     struct('tid', 4, 'p1','OB',   'p2',[])),...
    GTask('R2_PlacePrt_1', struct('tid', 2, 'p1',[],     'p2',[])),...
24  GTask('R2_Move_4',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
    ... %PM
26  R(    'R1'),...
    R(    'R2'),...
28  RT(   'RT'),...
```

```
    ... %IM
30  INTF( 'INTF1',        struct('rid', 1)),...
    INTF( 'INTF2',        struct('rid', 2)),...
32  RTC(  'RTC')
    });
34
  RM.set_Zid({
36  ... %CM
    'NoOp'          'NXT'    'R1_IdPrt_1'    'BEG';...
38  'NoOp'          'NXT'    'R2_IdPrt_1'    'BEG';...
    ... %CM (R1)
40  'R1_IdPrt_1'    'NXT'    'R1_Lock_1'     'BEG';...
    'R1_Lock_1'     'NXT'    'R1_Move_1'     'BEG';...
42  'R1_Move_1'     'NXT'    'R1_PickPrt_1'  'BEG';...
    'R1_PickPrt_1'  'NXT'    'R1_Move_2'     'BEG';...
44  'R1_Move_2'     'NXT'    'R1_UnLock_1'   'BEG';...
    'R1_UnLock_1'   'NXT'    'R1_Move_3'     'BEG';...
46  'R1_Move_3'     'NXT'    'R1_PlacePrt_1' 'BEG';...
    'R1_PlacePrt_1' 'NXT'    'R1_Move_4'     'BEG';...
48  'R1_Move_4'     'NXT'    'R1_IdPrt_1'    'BEG';...
    'R1_IdPrt_1'    'CMD'    'R1'            'CM_CMD';...
50  'R1_Lock_1'     'CMD'    'R1'            'CM_CMD';...
    'R1_Move_1'     'CMD'    'R1'            'CM_CMD';...
52  'R1_PickPrt_1'  'CMD'    'R1'            'CM_CMD';...
    'R1_Move_2'     'CMD'    'R1'            'CM_CMD';...
54  'R1_UnLock_1'   'CMD'    'R1'            'CM_CMD';...
    'R1_Move_3'     'CMD'    'R1'            'CM_CMD';...
56  'R1_PlacePrt_1' 'CMD'    'R1'            'CM_CMD';...
    'R1_Move_4'     'CMD'    'R1'            'CM_CMD';...
58  ... %CM (R2)
    'R2_IdPrt_1'    'NXT'    'R2_Lock_1'     'BEG';...
60  'R2_Lock_1'     'NXT'    'R2_Move_1'     'BEG';...
    'R2_Move_1'     'NXT'    'R2_PickPrt_1'  'BEG';...
62  'R2_PickPrt_1'  'NXT'    'R2_Move_2'     'BEG';...
    'R2_Move_2'     'NXT'    'R2_UnLock_1'   'BEG';...
64  'R2_UnLock_1'   'NXT'    'R2_Move_3'     'BEG';...
    'R2_Move_3'     'NXT'    'R2_PlacePrt_1' 'BEG';...
66  'R2_PlacePrt_1' 'NXT'    'R2_Move_4'     'BEG';...
    'R2_Move_4'     'NXT'    'R2_IdPrt_1'    'BEG';...
68  'R2_IdPrt_1'    'CMD'    'R2'            'CM_CMD';...
    'R2_Lock_1'     'CMD'    'R2'            'CM_CMD';...
70  'R2_Move_1'     'CMD'    'R2'            'CM_CMD';...
    'R2_PickPrt_1'  'CMD'    'R2'            'CM_CMD';...
72  'R2_Move_2'     'CMD'    'R2'            'CM_CMD';...
    'R2_UnLock_1'   'CMD'    'R2'            'CM_CMD';...
74  'R2_Move_3'     'CMD'    'R2'            'CM_CMD';...
    'R2_PlacePrt_1' 'CMD'    'R2'            'CM_CMD';...
76  'R2_Move_4'     'CMD'    'R2'            'CM_CMD';...
    ... %PM (R1)
78  'R1'            'CM_STS' 'R1_IdPrt_1'    'STS';...
    'R1'            'CM_STS' 'R1_Lock_1'     'STS';...
80  'R1'            'CM_STS' 'R1_Move_1'     'STS';...
    'R1'            'CM_STS' 'R1_PickPrt_1'  'STS';...
82  'R1'            'CM_STS' 'R1_Move_2'     'STS';...
    'R1'            'CM_STS' 'R1_UnLock_1'   'STS';...
84  'R1'            'CM_STS' 'R1_Move_3'     'STS';...
    'R1'            'CM_STS' 'R1_PlacePrt_1' 'STS';...
86  'R1'            'CM_STS' 'R1_Move_4'     'STS';...
```

```
   ... %PM (R2)
88 'R2'           'CM_STS' 'R2_IdPrt_1'    'STS';...
   'R2'           'CM_STS' 'R2_Lock_1'     'STS';...
90 'R2'           'CM_STS' 'R2_Move_1'     'STS';...
   'R2'           'CM_STS' 'R2_PickPrt_1'  'STS';...
92 'R2'           'CM_STS' 'R2_Move_2'     'STS';...
   'R2'           'CM_STS' 'R2_UnLock_1'   'STS';...
94 'R2'           'CM_STS' 'R2_Move_3'     'STS';...
   'R2'           'CM_STS' 'R2_PlacePrt_1' 'STS';...
96 'R2'           'CM_STS' 'R2_Move_4'     'STS';...
   ... %PM (R1 <> RT <> R2)
98 'R1'          'RT_OUT'  'RT'            'R1_IN';...
   'RT'          'R1_OUT'  'R1'            'RT_IN';...
100 'R2'         'RT_OUT'  'RT'            'R2_IN';...
   'RT'          'R2_OUT'  'R2'            'RT_IN';...
102 ... %IM (R1)
   'R1'          'IM_CMD'  'INTF1'         'CMD';...
104 'INTF1'      'STS'     'R1'            'IM_STS';...
   ... %IM (R2)
106 'R2'         'IM_CMD'  'INTF2'         'CMD';...
   'INTF2'       'STS'     'R2'            'IM_STS'
108 })
```

Listing G.11: Coupled-PDEVS RM zur Interaktionsklasse 4

```
   RM = coupled('RM');
2  RM.addcomponents({...
     ... %CM
4    NoOp( 'NoOp'),...
     ... %CM (R1)
6    IdPrt('R1_IdPrt_1',    struct('tid', 3, 'p1','IB',   'p2','A')),...
     GTask('R1_Lock_1',     struct('tid',-1, 'p1','IB',   'p2',[])),...
8    GTask('R1_Move_1',     struct('tid', 4, 'p1','Pos',  'p2',[])),...
     GTask('R1_PickPrt_1',  struct('tid', 1, 'p1',[],     'p2',[])),...
10   GTask('R1_Move_2',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
     GTask('R1_UnLock_1',   struct('tid',-2, 'p1','IB',   'p2',[])),...
12   GTask('R1_Move_3',     struct('tid', 4, 'p1','OB',   'p2',[])),...
     GTask('R1_PlacePrt_1', struct('tid', 2, 'p1',[],     'p2',[])),...
14   GTask('R1_Move_4',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
     IdPrt('R1_IdPrt_2',    struct('tid', 3, 'p1','IB',   'p2','B')),...
16   GTask('R1_Move_5',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
     GTask('R1_JoinRT_1',   struct('tid',-3, 'p1',[],     'p2',[])),...
18   ... %CM (R2)
     IdPrt('R2_IdPrt_1',    struct('tid', 3, 'p1','IB',   'p2','B')),...
20   GTask('R2_Lock_1',     struct('tid',-1, 'p1','IB',   'p2',[])),...
     GTask('R2_Move_1',     struct('tid', 4, 'p1','Pos',  'p2',[])),...
22   GTask('R2_PickPrt_1',  struct('tid', 1, 'p1',[],     'p2',[])),....
     GTask('R2_Move_2',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
24   GTask('R2_UnLock_1',   struct('tid',-2, 'p1','IB',   'p2',[])),...
     GTask('R2_Move_3',     struct('tid', 4, 'p1','OB',   'p2',[])),...
26   GTask('R2_PlacePrt_1', struct('tid', 2, 'p1',[],     'p2',[])),...
     GTask('R2_Move_4',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
28   IdPrt('R2_IdPrt_2',    struct('tid', 3, 'p1','IB',   'p2','B')),...
     GTask('R2_Move_5',     struct('tid', 4, 'p1','SPos', 'p2',[])),...
30   GTask('R2_JoinRT_1',   struct('tid',-3, 'p1',[],     'p2',[])),...
     ... %CM (RT)
32   GTask('RT_DefineRT',   struct('tid',-4, 'p1','R1',   'p2','R2')),...
     GTask('RT_Wait4RT',    struct('tid',-5, 'p1',[],     'p2',[])),...
```

```
34    GTask('RT_Move_1',      struct('tid', 4, 'p1','Pos',   'p2',[])),...
      GTask('RT_PickPrt_1',   struct('tid', 1, 'p1',[],      'p2',[])),...
36    GTask('RT_Move_2',      struct('tid', 4, 'p1','OB',    'p2',[])),...
      GTask('RT_PlacePrt_1',  struct('tid', 2, 'p1',[],      'p2',[])),...
38    GTask('RT_Move_3',      struct('tid', 4, 'p1','SPos',  'p2',[])),...
      GTask('RT_ExitRT',      struct('tid',-6, 'p1',[],      'p2',[])),...
40    ... %PM
      R(    'R1'),...
42    R(    'R2'),...
      RT(   'RT'),...
44    ... %IM
      INTF( 'INTF1',          struct('rid', 1)),...
46    INTF( 'INTF2',          struct('rid', 2)),...
      RTC(  'RTC')
48    });

50  RM.set_Zid({
      ... %CM
52    'NoOp'            'NXT'     'R1_IdPrt_1'      'BEG';...
      'NoOp'            'NXT'     'R2_IdPrt_1'      'BEG';...
54    'NoOp'            'NXT'     'RT_DefineRT'     'BEG';...
      ... %CM (R1)
56    'R1_IdPrt_1'      'NXT'     'R1_Lock_1'       'BEG';...
      'R1_Lock_1'       'NXT'     'R1_Move_1'       'BEG';...
58    'R1_Move_1'       'NXT'     'R1_PickPrt_1'    'BEG';...
      'R1_PickPrt_1'    'NXT'     'R1_Move_2'       'BEG';...
60    'R1_Move_2'       'NXT'     'R1_UnLock_1'     'BEG';...
      'R1_UnLock_1'     'NXT'     'R1_Move_3'       'BEG';...
62    'R1_Move_3'       'NXT'     'R1_PlacePrt_1'   'BEG';...
      'R1_PlacePrt_1'   'NXT'     'R1_Move_4'       'BEG';...
64    'R1_Move_4'       'NXT'     'R1_IdPrt_1'      'BEG';...
      'R1_IdPrt_1'      'NONE'    'R1_IdPrt_2'      'BEG';...
66    'R1_IdPrt_2'      'NXT'     'R1_Move_5'       'BEG';...
      'R1_IdPrt_2'      'NONE'    'R1_IdPrt_1'      'BEG';...
68    'R1_Move_5'       'NXT'     'R1_JoinRT_1'     'BEG';...
      'R1_JoinRT_1'     'NXT'     'R1_IdPrt_1'      'BEG';...
70    'R1_IdPrt_1'      'CMD'     'R1'              'CM_CMD';...
      'R1_Lock_1'       'CMD'     'R1'              'CM_CMD';...
72    'R1_Move_1'       'CMD'     'R1'              'CM_CMD';...
      'R1_PickPrt_1'    'CMD'     'R1'              'CM_CMD';...
74    'R1_Move_2'       'CMD'     'R1'              'CM_CMD';...
      'R1_UnLock_1'     'CMD'     'R1'              'CM_CMD';...
76    'R1_Move_3'       'CMD'     'R1'              'CM_CMD';...
      'R1_PlacePrt_1'   'CMD'     'R1'              'CM_CMD';...
78    'R1_Move_4'       'CMD'     'R1'              'CM_CMD';...
      'R1_IdPrt_2'      'CMD'     'R1'              'CM_CMD';...
80    'R1_Move_5'       'CMD'     'R1'              'CM_CMD';...
      'R1_JoinRT_1'     'CMD'     'R1'              'CM_CMD';...
82    ... %CM (R2)
      'R2_IdPrt_1'      'NXT'     'R2_Lock_1'       'BEG';...
84    'R2_Lock_1'       'NXT'     'R2_Move_1'       'BEG';...
      'R2_Move_1'       'NXT'     'R2_PickPrt_1'    'BEG';...
86    'R2_PickPrt_1'    'NXT'     'R2_Move_2'       'BEG';...
      'R2_Move_2'       'NXT'     'R2_UnLock_1'     'BEG';...
88    'R2_UnLock_1'     'NXT'     'R2_Move_3'       'BEG';...
      'R2_Move_3'       'NXT'     'R2_PlacePrt_1'   'BEG';...
90    'R2_PlacePrt_1'   'NXT'     'R2_Move_4'       'BEG';...
      'R2_Move_4'       'NXT'     'R2_IdPrt_1'      'BEG';...
```

```
92   'R2_IdPrt_1'    'NONE'   'R2_IdPrt_2'    'BEG';...
     'R2_IdPrt_2'    'NXT'    'R2_Move_5'     'BEG';...
94   'R2_IdPrt_2'    'NONE'   'R2_IdPrt_1'    'BEG';...
     'R2_Move_5'     'NXT'    'R2_JoinRT_1'   'BEG';...
96   'R2_JoinRT_1'   'NXT'    'R2_IdPrt_1'    'BEG';...
     'R2_IdPrt_1'    'CMD'    'R2'            'CM_CMD';...
98   'R2_Lock_1'     'CMD'    'R2'            'CM_CMD';...
     'R2_Move_1'     'CMD'    'R2'            'CM_CMD';...
100  'R2_PickPrt_1'  'CMD'    'R2'            'CM_CMD';...
     'R2_Move_2'     'CMD'    'R2'            'CM_CMD';...
102  'R2_UnLock_1'   'CMD'    'R2'            'CM_CMD';...
     'R2_Move_3'     'CMD'    'R2'            'CM_CMD';...
104  'R2_PlacePrt_1' 'CMD'    'R2'            'CM_CMD';...
     'R2_Move_4'     'CMD'    'R2'            'CM_CMD';...
106  'R2_IdPrt_2'    'CMD'    'R2'            'CM_CMD';...
     'R2_Move_5'     'CMD'    'R2'            'CM_CMD';...
108  'R2_JoinRT_1'   'CMD'    'R2'            'CM_CMD';...
     ... %CM (RT)
110  'RT_DefineRT'   'NXT'    'RT_Wait4RT'    'BEG';...
     'RT_Wait4RT'    'NXT'    'RT_Move_1'     'BEG';...
112  'RT_Move_1'     'NXT'    'RT_PickPrt_1'  'BEG';...
     'RT_PickPrt_1'  'NXT'    'RT_Move_2'     'BEG';...
114  'RT_Move_2'     'NXT'    'RT_PlacePrt_1' 'BEG';...
     'RT_PlacePrt_1' 'NXT'    'RT_Move_3'     'BEG';...
116  'RT_Move_3'     'NXT'    'RT_ExitRT'     'BEG';...
     'RT_ExitRT'     'NXT'    'RT_DefineRT'   'BEG';...
118  'RT_DefineRT'   'CMD'    'RT'            'CM_CMD';...
     'RT_Wait4RT'    'CMD'    'RT'            'CM_CMD';...
120  'RT_Move_1'     'CMD'    'RT'            'CM_CMD';...
     'RT_PickPrt_1'  'CMD'    'RT'            'CM_CMD';...
122  'RT_Move_2'     'CMD'    'RT'            'CM_CMD';...
     'RT_PlacePrt_1' 'CMD'    'RT'            'CM_CMD';...
124  'RT_Move_3'     'CMD'    'RT'            'CM_CMD';...
     'RT_ExitRT'     'CMD'    'RT'            'CM_CMD';...
126  ... %PM (R1)
     'R1'           'CM_STS'  'R1_IdPrt_1'    'STS';...
128  'R1'           'CM_STS'  'R1_Lock_1'     'STS';...
     'R1'           'CM_STS'  'R1_Move_1'     'STS';...
130  'R1'           'CM_STS'  'R1_PickPrt_1'  'STS';...
     'R1'           'CM_STS'  'R1_Move_2'     'STS';...
132  'R1'           'CM_STS'  'R1_UnLock_1'   'STS';...
     'R1'           'CM_STS'  'R1_Move_3'     'STS';...
134  'R1'           'CM_STS'  'R1_PlacePrt_1' 'STS';...
     'R1'           'CM_STS'  'R1_Move_4'     'STS';...
136  'R1'           'CM_STS'  'R1_IdPrt_2'    'STS';...
     'R1'           'CM_STS'  'R1_Move_5'     'STS';...
138  'R1'           'CM_STS'  'R1_JoinRT_1'   'STS';...
     ... %PM (R2)
140  'R2'           'CM_STS'  'R2_IdPrt_1'    'STS';...
     'R2'           'CM_STS'  'R2_Lock_1'     'STS';...
142  'R2'           'CM_STS'  'R2_Move_1'     'STS';...
     'R2'           'CM_STS'  'R2_PickPrt_1'  'STS';...
144  'R2'           'CM_STS'  'R2_Move_2'     'STS';...
     'R2'           'CM_STS'  'R2_UnLock_1'   'STS';...
146  'R2'           'CM_STS'  'R2_Move_3'     'STS';...
     'R2'           'CM_STS'  'R2_PlacePrt_1' 'STS';...
148  'R2'           'CM_STS'  'R2_Move_4'     'STS';...
     'R2'           'CM_STS'  'R2_IdPrt_2'    'STS';...
```

```
150  'R2'            'CM_STS'  'R2_Move_5'     'STS';...
     'R2'            'CM_STS'  'R2_JoinRT_1'   'STS';...
152  ... %PM (RT)
     'RT'            'CM_STS'  'RT_DefineRT'   'STS';...
154  'RT'            'CM_STS'  'RT_Wait4RT'    'STS';...
     'RT'            'CM_STS'  'RT_Move_1'     'STS';...
156  'RT'            'CM_STS'  'RT_PickPrt_1'  'STS';...
     'RT'            'CM_STS'  'RT_Move_2'     'STS';...
158  'RT'            'CM_STS'  'RT_PlacePrt_1' 'STS';...
     'RT'            'CM_STS'  'RT_Move_3'     'STS';...
160  'RT'            'CM_STS'  'RT_ExitRT'     'STS';...
     ... %PM (R1 <> RT <> R2)
162  'R1'            'RT_OUT' 'RT'             'R1_IN';...
     'RT'            'R1_OUT' 'R1'             'RT_IN';...
164  'R2'            'RT_OUT' 'RT'             'R2_IN';...
     'RT'            'R2_OUT' 'R2'             'RT_IN';...
166  ... %IM (R1)
     'R1'            'IM_CMD' 'INTF1'          'CMD';...
168  'INTF1'         'STS'    'R1'             'IM_STS';...
     ... %IM (R2)
170  'R2'            'IM_CMD' 'INTF2'          'CMD';...
     'INTF2'         'STS'    'R2'             'IM_STS'
172  })
```

SNE SIMULATION NOTES EUROPE

Simulation Notes Europe (**SNE**) provides an international, high-quality forum for presentation of new ideas and approaches in simulation - from modelling to experiment analysis, from implementation to verification, from validation to identification, from numerics to visualisation - in context of the simulation process.

SNE seeks to serve scientists, researchers, developers and users of the simulation process across a variety of theoretical and applied fields in pursuit of novel ideas in simulation and to enable the exchange of experience and knowledge through descriptions of specific applications. **SNE** puts special emphasis on the overall view in simulation, and on comparative investigations, as benchmarks and comparisons in methodology and application. Additionally, **SNE** welcomes also contributions in education in / for / with simulation.

SNE is the official membership journal of EUROSIM, the federation of European simulation societies and simulation groups, so **SNE** is open for postconference publication of contributions to conferences of the EUROSIM societies, and for special issues organized by EUROSIM societies.

SNE, primarily an electronic journal, follows an open access strategy, with free download in basic layout. Members of EUROSIM societies are entitled to download SNE in an elaborate and extended layout. Print **SNE** is available for specific groups of EUROSIM societies, and as print-on-demand from TU Verlag, TU Wien (www.tuverlag.at).

www.sne-journal.org

Über den Autor

Birger Freymann, geboren 1989 in Ludwigslust, studierte von 2008 bis 2014 Maschinenbau an der Hochschule Wismar. Während der Masterarbeit beschäftigte er sich mit Problemstellungen der Modellbildung, Simulation und Automation ereignisdiskreter Steuerungen. Im Anschluss an sein Studium arbeitete er als wissenschaftlicher Mitarbeiter in der Forschungsgruppe Computational Engineering and Automation (CEA) an der Hochschule Wismar. Motiviert aus dieser Tätigkeit folgte ein kooperatives Promotionsvorhaben mit der TU Clausthal zum Thema aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS. Die Ergebnisse dieser Arbeit werden in diesem Band präsentiert. Momentan ist der Autor als Technologiereferent bei der DB Fahrzeuginstandhaltung tätig.

Über dieses Buch

Die Integration und Vernetzung von Industrierobotern in komplexen automatisierten Systemen erfordert eine verstärkte Realisierung modellbasierter Methoden zur herstellerunabhängigen und applikationsübergreifenden Entwicklung von Roboteranwendungen. In dieser Arbeit wird ein Ansatz zur durchgängigen, modellbasierten und herstellerunabhängigen Steuerungsentwicklung für Roboterteams mit industriellen Knickarmrobotern entwickelt. Aufbauend auf dem Simulation-Based-Control (SBC)-Ansatz und dem Task-Oriented-Control (TOC)-Ansatz werden Entwicklungsmethoden aus dem Bereich von Single-Robotersystemen (SRS) auf Multi-Robotersysteme (MRS) übertragen. Es werden mögliche Interaktionen zwischen Robotern untersucht und darauf aufbauend Interaktionsklassen definiert. Zur Umsetzung einer durchgängigen Steuerungsentwicklung wird der Discrete-Event-System-Specification (DEVS)-Formalismus diskutiert und es werden Erweiterungen zur Echtzeit- und Prozessanbindung untersucht.

About the Series

The ASIM series Advances in Simulation / Fortschrittsberichte Simulation presents new and recent approaches, methods, and applications in modelling and simulation. The topics may range from theory and foundations via simulation techniques and simulation concepts to applications. As the spectrum of simulation techniques and applications is increasing, books in these series present classical techniques and applications in engineering, natural sciences, biology, physiology, production and logistics, and business administration, upcoming simulation applications in social sciences, media, data management, networking, and complex systems, and upcoming new simulation techniques as agent-based simulation, co-simulation, and deep learning, etc.

The series puts emphasis on monographs with special character, as PhD theses, habilitation treatises, project reports and overviews on scientific projects. ASIM - Arbeitsgemeinschaft Simulation, the German Simulation Society (part of GI - Gesellschaft für Informatik) has founded the series Advances in Simulation / Fortschrittsberichte Simulation together with ARGESIM Publisher Vienna in order to provide to the international simulation community a quick and cost-efficient print and e-book series with open access.

ISBN print
ISBN 978-3-903311-20-6
TU Verlag, Vienna, 2022
www.tuverlag.at

ISBN ebook
ISBN 978-3-903347-40-3

DOI ID
DOI 10.11128/fbs.40

ARGESIM Publisher, Vienna, 2022
www.argesim.org