

MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Dipl.-Wirt.-Inf.
Arne Haber
aus Wolfsburg

Berichter: Universitätsprofessor Dr. rer. nat. Bernhard Rumpe
 Universitätsprofessorin Dr.-Ing. Ina Schaefer

Tag der mündlichen Prüfung: 22. März 2016

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Aachener Informatik-Berichte, Software Engineering

herausgegeben von
Prof. Dr. rer. nat. Bernhard Rumpe
Software Engineering
RWTH Aachen University

Band 24

Arne Haber

MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems

Shaker Verlag
Aachen 2016

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Zugl.: D 82 (Diss. RWTH Aachen University, 2016)

Copyright Shaker Verlag 2016

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8440-4697-7

ISSN 1869-9170

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen

Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9

Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

Software architecture is the essence of a system and determines important functional as well as non-functional properties [BCK03]. It decomposes the system into small, manageable sub-systems respectively components that interact over well defined interfaces. Formal architecture description languages (ADLs) offer great potential to model and analyse the architecture of a system, predict the overall performance of a system using simulations, and even allow to automatically generate parts of the implementation.

Nevertheless, ADLs are rather not used in industrial practice since several problems of architectural modeling hinder to exploit their potential to the full extend. Either an ADL is too general to provide enough information for formal analyses, or it is tailored well to a specific domain and development process, so it cannot be easily applied to another context. Beside language barriers caused by uncommon languages, most ADLs are regarded to be complex and heavyweight [MLM⁺13]. Good modeling tools are missing [Pan10] and existing tools cannot be easily integrated into existing tool chains [WH05]. Also, incremental modeling and model reuse is most often not supported.

This thesis elaborates the design of an ADL that copes with these impediments of architectural modeling in practice. Therefore, the design of a lightweight and easy to learn ADL is derived which also provides well defined extension points to be adapted to a certain domain or development process. Controlled reuse of architectural models is explored. Furthermore, it is investigated how architectural modeling can be enriched with agile development methods to support incremental modeling and the validation of system architectures.

Therefore, a detailed set of requirements for architectural modeling and the simulation of system architectures is defined. Based on these requirements, MontiArc, a concrete ADL to model logical architectures of distributed, interactive systems, is derived. The language is based on the mathematical FOCUS [BS01] framework, which allows to simulate modeled systems in an event-based style. Code generators and a simulation framework provide means to continuously refine and test architectural models.

The language and the corresponding tools provide extension points to easily add new features to the language or even adapt it to a new domain. For this purpose, a corresponding language extension method is presented to extend the syntax, language processing tools, and code generators of the ADL. Furthermore, a lightweight model library concept is presented which allows to develop and reuse component models and their implementation in a controlled and transparent way. The developed language, the simulator, and the language extension techniques have been examined in several case studies which either used or extended MontiArc.

Danksagung

An dieser Stelle möchte ich mich herzlich bei den Menschen bedanken, die zur Fertigstellung und zum Gelingen dieser Dissertation beigetragen haben.

Besonderer Dank gilt meinem Doktorvater Prof. Dr. Bernhard Rumpel für die Betreuung dieser Dissertation. Durch zahlreiche konstruktive Diskussionen mit Bernhard und seine wertvollen Ratschläge hat er entscheidend zum Gelingen der Promotion beigetragen. Ebenfalls bedanken möchte ich mich für die Möglichkeit in vielen Industrie- und Forschungsprojekten Erfahrungen außerhalb des akademischen Elfenbeinturms zu sammeln. Diese Erfahrungen haben maßgeblich zu meinem weiteren Werdegang beigetragen.

Weiterer Dank gebührt Prof. Dr.-Ing. Ina Scheafer für die gemeinsamen spannenden Arbeiten und Projekte im Bereich der Varianten- und Deltamodellierung. Über ihre Bereitschaft meine Dissertation als Zweitgutachterin zu betreuen, habe ich mich besonders gefreut. Prof. Dr. Ir. Joost-Pieter Katoen möchte ich für die Leitung der Promotionskommission sowie für die Abnahme der Prüfung in der theoretischen Informatik danken. Herrn Prof. Dr. Stefan Decker danke ich für die Bereitschaft mich in der praktischen Informatik zu prüfen.

Natürlich möchte ich mich sehr herzlich bei meinen Kollegen bedanken, mit denen ich die Zeit am Lehrstuhl, beim Kickerspielen oder bei abendlicher Zerstreuung verbracht habe. Ihr habt dazu beigetragen, dass Aachen ein Stückchen Heimat für mich wurde, auch wenn ich dem Karneval nie etwas abgewinnen konnte. Für das Korrekturlesen früher Fassungen der Dissertation bedanke ich mich bei Lars Hermerschmidt, Andreas Horst, Thomas Kurpick, Markus Look, Klaus Müller, Pedram Mir Seyed Nazari, Antonio Navarro Pérez und Andreas Wortmann. Da sich die entsprechenden Unterlagen bereits im Keller befinden (wer eine Dissertation geschrieben hat, kann das sicherlich nachvollziehen), habe ich hier hoffentlich niemanden vergessen. Für das Anhören des Probevortrags und hilfreiches Feedback möchte ich Achim Lindt, Markus Look und Andreas Wortmann danken.

Dr. Jan Oliver Ringert danke ich für die gemeinsame Arbeit am MontiArc Sprachdesign und dem technischen Bericht. Darüber hinaus nochmal vielen Dank an die "Technologiesau" Antonio und Andreas H. für euren stetigen Kampf für Maven und die Unterstützung bei der Umstellung von MontiArc auf dieses Buildsystem. Dr. Holger Krahn, Dr. Steven Völkel und Dr. Martin Schindler gilt mein Dank für ihre Arbeiten an MontiCore, auf deren Basis meine Arbeit aufgesetzt hat. Marita Breuer und Galina Volkova danke ich für den technischen MontiCore-Support. Bei Sylvia Gunder und (Dr.) Holger Rendel bedanke ich mich für die Unterstützung bei der organisatorischen Vorbereitung von Holgers und meiner Promotionsprüfungen, die am selben Tag durchgeführt wurden.

Bedanken möchte ich mich ebenfalls bei den vielen Studenten, die mich in ihrer Abschlussarbeit oder als Hiwi bei der Implementierung in und um MontiArc oder bei Projekten, unterstützt haben. Hierzu gehören: Markus Arndt, Paul Chomicz, Christoph Hommelsheim, Tim Ix, Oliver

Kautz, Alexander Kogaj, Thomas Kutz, Juha Veikko Luttamus, mein späterer Kollege Pedram Mir Seyed Nazari, Rajeevan Rabindran, Sebastian Roidl, Stefan Schubert und Minh Quan Tran. Bei Claas Oppitz bedanke ich mich für das Design und die Erstellung des MontiArc Logos.

Abschließend möchte ich meinen Eltern und meiner Familie für die Unterstützung auf meinem Lebensweg danken. Ihr habt mir das Studium und die anschließende Promotion ermöglicht und mich stets in meinem Vorhaben bestärkt. Besonderer Dank gebührt natürlich Nathalie, die mich nicht nur unterstützt, sondern auch meine Laune bei der Erstellung dieser Arbeit aufgeheitert hat. Danke, dass Du gemeinsam mit mir für die Zeit der Promotion nach Aachen gezogen bist. Ohne Dich wäre diese Arbeit nicht möglich gewesen.

Braunschweig, Juli 2016
Arne Haber

Contents

1	Introduction	1
1.1	Terms and Definitions	2
1.2	Motivation	4
1.3	Context	5
1.4	Objectives	6
1.5	Main Results	8
1.6	Thesis' Structure	8
2	Requirements for Architectural Modeling and Simulation	11
2.1	Requirements for Architectural Modeling	11
2.2	Simulation Requirements	15
2.3	Currently existing architecture description languages (ADLs)	17
2.3.1	AADL	18
2.3.2	Acme and xADL	22
2.3.3	AutoFocus 3	24
2.3.4	ArchJava and Java/A	26
2.3.5	Ptolemy II	28
2.3.6	UML and SysML	30
2.3.7	Summary	33
3	MontiArc ADL	39
3.1	A MontiArc Example	40
3.2	Basic Architectural Model Elements	42
3.2.1	Component Type Definition	42
3.2.2	Component Interface	44
3.2.3	Architectural Configuration	45
3.3	Advanced Architectural Model Elements	47
3.3.1	Component Timing	47
3.3.2	Autoconnect	47
3.3.3	Autoinstantiate	48
3.3.4	Constraints	48
3.4	MontiArc Language Definition	49
3.4.1	Foundations: MontiCore 3	50
3.4.2	Language Structure	52
3.4.3	Architecture Diagram Grammar Walk-Through	53
3.4.4	MontiArc Grammar Walk-Through	56

3.5	Context Conditions	57
3.5.1	General Conditions	57
3.5.2	Connections	60
3.5.3	Referential Integrity	62
3.5.4	Conventions	70
3.5.5	Code Generation	73
3.6	AADL Compatibility	74
3.6.1	AADL Components	74
3.6.2	AADL Interfaces	77
3.6.3	AADL Architectural Configuration	78
3.6.4	Further AADL Modeling Elements	80
3.6.5	Summary	81
4	Simulating MontiArc Models	85
4.1	Foundations for the Simulator	85
4.2	Runtime Environment	89
4.2.1	Intended Object Structure @Runtime	89
4.2.2	Simulation Runtime Environment	93
4.3	Scheduling	96
4.3.1	Scheduling of Data Messages	98
4.3.2	Scheduling of Ticks	100
4.3.3	Scheduling already Scheduled or Blocked Ports	102
4.3.4	Waking up Ports	103
4.4	Timing Classification	105
4.4.1	Instant Timing	106
4.4.2	Delayed Timing	108
4.4.3	Untimed	108
4.4.4	Synchronous Timing	109
4.4.5	Causal Synchronous Timing	110
4.4.6	Timing Domain Overview	110
4.5	Optimization and Runtime Measurement	110
4.5.1	Simple Round Robin Scheduling	111
4.5.2	Further Optimization potential	114
4.5.3	Scheduler Variants	114
4.5.4	Comparison Setup	116
4.5.5	Results	117
4.5.6	Discussion of the Results	119
4.6	Technical Design Decisions	121
5	Technical Realization of MontiArc	125
5.1	Model Processing	125
5.2	Symbol Table	129
5.2.1	Foundations	130
5.2.2	Symbol Table Construction	131

5.2.3	MontiArc Symbol Table: Namespace Hierarchy	133
5.2.4	MontiArc Symbol Table: Structure	135
5.2.5	MontiArc Symbol Table: Model Interfaces	138
5.3	Transformations	140
5.3.1	Pre Context-Condition Transformations	140
5.3.2	Pre Code Generation Transformations	144
5.3.3	Implementation	148
5.4	Generation of Simulation Code	149
5.4.1	Component Interfaces	150
5.4.2	Atomic Components	151
5.4.3	Decomposed Components	156
5.5	Atomic Component Behavior Implementation	160
5.5.1	Implementation	160
5.5.2	Integration of Handwritten Code	165
5.5.3	Components with Side-Effects	167
5.6	Reduction of Redundant Objects	169
5.6.1	Atomic Components with a Single Incoming Port	169
5.6.2	Reduction of ForwardPorts in Decomposed Components	171
5.6.3	Reuse of Tick Objects	172
5.7	MontiArc Tools	173
5.7.1	Command Line Interface	173
5.7.2	MontiArc Maven Plugin	176
5.7.3	Eclipse IDE	178

6 Tutorial: Development and Simulation of MontiArc Components 181

6.1	Getting Started	182
6.2	Illustrative Example - Alternating Bit Protocol	183
6.2.1	Requirements	184
6.2.2	Example Setup	185
6.2.3	Modeling	186
6.3	Behavior Implementation	191
6.3.1	Behavior Implementation in Java	191
6.3.2	Native Behavior Implementation	192
6.4	Validation of MontiArc Models	196
6.4.1	Model-Based Black-box Tests	196
6.4.2	White-box Testing of Decomposed Models	200
6.5	Generalize Components	205
6.6	Optimization Testing	209
6.7	Documentation of MontiArc Models	212
6.7.1	Enabling the Documentation Generator	212
6.7.2	Document Components	213
6.7.3	Index Page Design	216
6.7.4	Package Documentation	217

6.8	MontiArc Libraries	217
6.8.1	Structure of a Model Library	217
6.8.2	Predefined Libraries	218
6.8.3	Creating a Library	219
6.8.4	Using a Library	219
6.9	Distributed Simulation	220
7	MontiArc Extension Method	225
7.1	Model Processing Extension	226
7.1.1	Add Execution Unit	227
7.1.2	Add Transformation	229
7.2	Simulation Extension	230
7.2.1	Handle Extended Syntax	230
7.2.2	Add Feature	230
7.2.3	Extend Scheduling	232
7.2.4	Code Generator Extension	232
7.3	Language Extension	234
7.3.1	Syntax Extension	234
7.3.2	Symbol Table Extension	240
8	Case Studies Using MontiArc	245
8.1	Overview	245
8.2	Modeling and Simulation of the TCP/IP Stack	246
8.2.1	The TCP/IP Stack - An Introduction	246
8.2.2	TCP/IP Stack Layers in MontiArc	247
8.2.3	Conclusion	250
8.3	FlexRay Communication Simulation Using MontiArc	250
8.3.1	FlexRay Introduction	251
8.3.2	The Running Example	253
8.3.3	Deployment and FlexRay components	253
8.3.4	The MontiArc FlexRay Generator	254
8.3.5	Conclusion	256
9	Language Extension Case Studies	257
9.1	Overview	257
9.2	AJava	259
9.2.1	Example	260
9.2.2	Language and Tool Extensions	262
9.2.3	Conclusion	264
9.3	MontiArcAutomaton	265
9.3.1	Example	265
9.3.2	Language Extensions	266
9.3.3	Conclusion	268

9.4	Process Network Simulation	269
9.4.1	Example	269
9.4.2	Discuss Language Extension	271
9.4.3	Conclusion	274
10	Discussion and Conclusion	275
10.1	Requirements for Architectural Modeling	275
10.1.1	<i>LRQ1</i> : Architectural Style	275
10.1.2	<i>LRQ2</i> : Usability	278
10.1.3	<i>LRQ3</i> : Reusability and Extensibility	279
10.1.4	<i>LRQ4</i> : Type System	279
10.1.5	<i>LRQ5</i> : Libraries	279
10.2	Simulation Requirements	280
10.2.1	<i>SRQ1</i> : Platform Independence	281
10.2.2	<i>SRQ2</i> : External Component Implementation	281
10.2.3	<i>SRQ3</i> : Mathematical Foundation	282
10.2.4	<i>SRQ4</i> : Component Timing Classification	283
10.2.5	<i>SRQ5</i> : Simulation Time	283
10.2.6	<i>SRQ6</i> : Distribution	283
10.2.7	<i>SRQ7</i> : Component Testing	284
10.2.8	<i>SRQ8</i> : Extensibility	285
10.2.9	<i>SRQ9</i> : Scheduler	285
10.2.10	<i>SRQ10</i> : Optimizations	286
10.3	Conclusion	286
A	Index of Abbreviations	293
B	Diagram and Listing Tags	295
C	Grammars	297
C.1	Architectural Diagrams Grammar	297
C.2	MontiArc Grammar	302
C.3	I/O Test Language Grammar	305
C.4	Process Network Simulation Grammar	312
D	AADL Examples	313
E	Tutorial Material	317
E.1	Implementations	317
E.2	I/O-Test Models	320
E.3	White-Box Tests	324
E.4	Generalized Components	328
E.5	Optimization Testing	329
E.6	Distributed Simulation	335

F	Language Extension Material	337
G	Curriculum Vitae	343
	Bibliography	345
	List of Figures	365
	Listings	369
	List of Tables	373