An Extensible Component & Connector Architecture Description Infrastructure for Multi-Platform Modeling

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

Diplom Informatiker Diplom-Wirtschaftsinformatiker
Andreas Wortmann
aus Joinville, Brasilien

Berichter: Universitätsprofessor Dr. Bernhard Rumpe
Professor Benoit Combemale, Ph. D.

Tag der mündlichen Prüfung: 12. Juli 2016

Aachener Informatik-Berichte, Software Engineering

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

Band 25

Andreas Wortmann

An Extensible Component & Connector Architecture Description Infrastructure for Multi-Platform Modeling

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)

Printed in Germany.

ISBN 978-3-8440-4724-0
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

I was born not knowing
and have had only a little time
to change that here and there.

Richard P. Feynman

Abstract

Efficient software engineering for complex systems requires abstraction, expertise from multiple domains, separation of concerns, and reuse. Domain experts are rarely software engineers and should be enabled to formulate solutions using their domain's vocabulary instead of general-purpose programming languages (GPLs). The successful integration of domain-specific languages (DSLs) into a software system requires a separation of concerns between domain issues and integration issues while retaining a loose enough coupling to support reusing a DSL in different contexts.

Component-based software engineering (CBSE) aims to increase software reuse and separation of concerns by encapsulating functionalities in components. This enables domain experts to develop solutions separated from integration concerns. Usually components are artifacts of GPLs, which gives rise to accidental complexities [FR07] and ties these to specific target platforms.

Model-driven engineering (MDE) abstracts from programming by lifting models to primary development artifacts. Models can be more abstract and better comprehensible by using domain vocabulary instead of a GPL. Furthermore, they can be platform-independent and translated into GPLs for different target platforms.

Component & connector (C&C) architecture description languages (ADLs) combine CBSE and MDE to enable composition of software architectures from component models. Such models define stable interfaces required to separate domain concerns from integration concerns. They can also employ the most appropriate DSLs to describe component behavior and support translation into GPL artifacts specific to different target platforms. Current research in MDE with ADLs focuses on structural modeling and requires component behavior either in terms of GPL artifacts or fixed component behavior languages. The former gives rise to accidental complexities, the latter demands that domain experts learn modeling languages foreign to their domain.

This thesis presents concepts for engineering complex software systems with exchangeable component behavior languages that enable contribution of domains experts using the most appropriate DSLs. The concepts are realized in a software architecture modeling infrastructure that comprises multiple modeling languages to develop applications based on C&C software architectures with exchangeable component behavior languages. It supports model-to-model transformations from platform-independent to platform-specific software architectures and compositional code generation. With this, it enables domain experts to (re-)use the most appropriate component behavior languages and facilitates composition of domain solutions through encapsulation in components.

It also enables reusing a single platform-independent software architecture with multiple platforms. To this effect, it combines results from software language engineering, model transformations, and code generator development to C&C ADLs.

The main contributions of this thesis are:

- Concepts to integrate domain-specific languages into component & connector software architectures to reduce accidental complexities, separate concerns, and facilitate their reuse.
- Methodical guidance to transform platform-independent into platform-specific architectures minor effort to increase reuse of components and architectures.
- Concepts of reusable compositional code generators for specific system aspects.
- A family of modeling languages to support architecture development with exchangeable behavior DSLs.
- A model-driven infrastructure, based on an extensible component & connector architecture description language that realizes these concepts.
- An evaluation of presented concepts in multiple contexts.

Employing these methodologies facilitates engineering of complex software systems by abstracting from programming issues, separating concerns, and reusing components, domain-specific languages, as well as code generators.

Kurzfassung

Die effiziente Softwareentwicklung komplexer System bedarf hoher Abstraktion, der Beteiligung von Experten aus verschiedenen Domänen, der Trennung von Belangen und eines hohen Grades an Wiederverwendung. Domänenexperten sind selten Softwareexperten und sollten daher befähigt werden Lösungen im Vokabular ihrer Domänen zu entwickeln. Um dies zu erreichen wurden in der modellgetriebenen Softwareentwicklung eine Vielzahl von domänenspezifischen Sprachen entwickelt Die erfolgreiche Integration domänenspezifischer Sprachen in Softwaresysteme bedarf einer angemessenen Trennung von Domänen- und Integrationsbelangen, wobei die Kopplung dieser Sprachen lose genug sein muss um deren Wiederverwendung in anderen Kontexten zu ermöglichen.

Komponentenbasierte Softwareentwicklung versucht die Wiederverwendung von Software Kapselung von Funktionalitäten in Komponenten zu erhöhen. Dies ermöglicht Domänenexperten Lösungen unabhängig von Integrationsbelangen zu entwickeln. In komponentenbasierter Softwareentwicklung werden Komponenten üblicherweise durch Allzweck-Programmiersprachen beschrieben. Dies führt zu "unbeabsichtigten Komplexitäten" [FR07], welche darin bestehen Programmierdetails anstelle von Domänenproblemen zu lösen und führt dazu, dass die Lösungen nur zu bestimmten Zielplattformen kompatibel sind.

Modellgetriebene Softwareentwicklung abstrahiert von Programmierdetails durch die Verwendung von Modellen als primäre Entwicklungsartefakte. Diesen können abstrakter und, durch Verwendung von Domänenvokabular, besser verständlich sein. Weiterhin können sie plattformunabhängig sein und durch Übersetzung in mehrere Allzweck-Programmiersprachen mit mehreren Plattformen wiederverwendet werden.

Komponenten und Konnektor Architekturbeschreibungssprachen kombinieren komponentenbasierte Softwareentwicklung mit modellgetriebener Softwareentwicklung zur Komposition von Softwarearchitekturen aus Komponentenmodellen. Diese Modelle verfügen über stabile Schnittstellen zur Trennung von Belangen, können Komponentenverhalten in angemessen domänenspezifischen Sprachen ausdrücken und ermöglichen eine automatisierte Übersetzung in plattformspezifische Artefakte. Gegenwärtige Forschung in der modellgetriebenen Entwicklung mit Architekturbeschreibungssprachen untersucht strukturelle Systemaspekte und erwartet Komponentenverhalten entweder in Form von Artefakten von Allzweck-Programmiersprachen oder in Form apriori festgelegter Modellierungssprachen. Ersteres führt zu unbeabsichtigten Komplexitäten, zweites erfordert dass Domänenexperten domänenfremde Sprachen lernen.

Diese Dissertation präsentiert Konzepte für die Entwicklung komplexer Softwaresysteme mit austauschbaren Komponentenverhaltenssprachen. Diese Konzepte sind in einer Infrastruktur zur Modellierung von Software-Architekturen realisiert welche mehrere Modellierungssprachen umfasst. Sie unterstützt die agile Einbettung angemessener Verhaltenssprachen, Modell-zu-Modell Transformationen von plattformunabhängigen zu plattformspezifischen Softwarearchitekturen, und kompositionale Code Generatoren. Dies ermöglicht Domänenexperten die angemessensten Komponentenverhaltenssprachen zu verwenden und eine einzige, plattformunabhängige, Softwarearchitektur mit verschiedenen Plattformen wieder zu verwenden. Um dies zu erreichen, kombiniert diese Infrastruktur Erkenntnisse aus der Entwicklung von Software-Sprachen, Modell-zu-Modell Transformationen, und aus der Code Generator Entwicklung mit Komponenten- und Konnektor Architekturbeschreibungssprachen.

Die wichtigsten Beiträge dieser Arbeit sind somit:

- Konzepte für die die Integration domänenspezifischer Sprachen in Komponenten- und Konnektor Softwarearchitekturen zur Reduktion unbeabsichtigter Komplexitäten, Trennung von Belangen und Erleichterung von deren Wiederverwendung.
- Eine Methodik zur Transformation plattformunabhängiger Softwarearchitekturen in plattformabhängige Softwarearchitekturen zur Erhöhung der Wiederverwendbarkeit von Komponenten und Architekturen.
- Ein Konzept zur Wiederverwendung kompositionaler Code Generatoren für bestimmte Systemaspekte.
- Eine Familie von Modellierungssprachen für die Architekturmodellierung mit austauschbaren Verhaltenssprachen.
- Eine modellgetriebene Werkzeugkette die diese Konzepte realisiert.
- Eine Evaluierung vorgestellter Konzepte in verschiedenen Kontexten.

Die Anwendung dieser Methodiken erleichtert die Entwicklung komplexer Softwaresysteme durch Abstraktion von Programmierungsdetails, durch eine gründliche Trennung von Belangen und durch Wiederverwendung von Komponenten, Architekturen, domänenspezifischen Sprachen und Code Generatoren.

Danksagung

Während meiner Promotion wurde ich von vielen Menschen unterstützt die damit zu dem Erfolg dieser Dissertation beigetragen haben und denen ich dafür sehr dankbar bin.

Größter Dank gebührt meinem Doktorvater Prof. Dr. Bernhard Rumpe, welcher diese Dissertation durch Unterstützung meiner Promotion erst möglich gemacht hat. Neben der Leitung einer großartigen Arbeitsgruppe haben viele fruchtbare Diskussionen mit ihm und seine Ratschläge diese Dissertation und meine wissenschaftlichen Tätigkeiten mitgeformt.

Ich danke außerdem Prof. Benoit Combemale, Ph.D., dem Zweitgutachter dieser Arbeit, für die sehr gute Zusammenarbeit und seine Unterstützung meiner Promotion. Mein Dank gebührt weiterhin Prof. Dr. Joost-Pieter Katoen dafür mein Promotionskomitee zu leiten und Prof. Dr. Ulrik Schroeder dafür in diesem mitzuarbeiten.

Außerdem danke ich den wundervollen Kollegen und Freunden am Lehrstuhl für Software Engineering der RWTH Aachen, welche die letzten fünf Jahre zu einer spannenden Zeit gemacht haben. Ohne das hervorragende Arbeitsklima, die fruchtbaren Anregungen und Diskussionen, und die gemeinsamen Arbeit an akademischen und organisatorischen Herausforderungen wäre diese Zeit kaum derart interessant gewesen. Besonders dankbar bin ich Dr. Jan Oliver Ringert, dessen Motivation, Unterstützung und Bereitschaft zu intensiven Diskussionen diese Dissertation stark beeinflusst haben. Besonderer Dank gilt auch Markus Look, welcher mich in vielen Dingen unterstützte und immer für aufschlussreiche Diskussionen da war, so wie Dr. Arne Haber, der geduldig für viele Fragen und Ideen zur Verfügung stand, und Andreas Horst, ohne dessen Hilfe manche Herausforderungen ungelöst wären. Weiterhin bedanke ich mich bei Robert Heim, dessen Unterstützung in verschiedenen Tätigkeiten den Endspurt der Promotion erleichtert hat.

Außerdem bedanke ich mich bei Prof. Dr. Christian Berger und Prof. Dr. Ulrike Thomas, welche mir die Möglichkeit gaben mich mit weiteren spannenden Forschungsfragen zu befassen und an Ihrer Erfahrung teilhaben ließen. Ich danke auch Dr. Stefan Schiffer, Dr. Martin Schindler und Prof. Dr. Christian Schlegel, deren Erfahrungen auf dem Weg zur Dissertation und darüber hinaus sehr hilfreich waren. Weiterer Dank gebührt Kai Adam, Marita Breuer, Arvid Butting, Angelika Fleck, Timo Greifenberg, Sylvia Gunder, Lars Hermerschmidt, Dr. Christoph Herrmann, Gabriele Heuschen, Katrin Hölldobler, Steffi Kaiser, Oliver Kautz, Dennis Kirch, Carsten Kolassa, Evgeny Kusmenko, Thomas Kurpick, Achim Lindt, Klaus Müller, Antonio Navarro Pérez, Jerome Pfeiffer, Prof. Dr. Manfred Nagl, Pedram Mir Seyed Nazari, Dr. Claas Pinkernell, Dimitri Plotnikov, Deni Raco, Holger Rendel, Dirk Reiss, Alexander Roth, Christoph

Schulze, Galina Volkova, Michael von Wenckstern, Dr. Ingo Weisemöller und Dr. Steven Völkel ohne deren Unterstützung in den vielen Herausforderungen dieser Promotion diese nicht derart möglich gewesen wäre. Nicht zuletzt danke ich meiner Familie, meiner Partnerin und meinen Freunden für ihre Unterstützung während dieser Zeit und für ihr Verständnis, wenn ich mich für die Arbeit rar gemacht habe. Besonders danke ich meinen Eltern für Ihre durchgängige Unterstützung aller Schritte die zu dieser Arbeit geführt haben.

Contents

1 Introduction and Motivation **1**
- 1.1 Motivation . . . 2
- 1.2 Main Goals and Results . . . 4
- 1.3 Thesis Organization . . . 5
- 1.4 Related Publications . . . 5

2 Preliminaries for Architecture Modeling **7**
- 2.1 Model-Based Software Engineering . . . 7
- 2.2 MontiCore . . . 11
  - 2.2.1 Symbol Table Framework . . . 14
  - 2.2.2 Language Integration Mechanisms . . . 15
  - 2.2.3 Code Generation Framework . . . 18
  - 2.2.4 Related Language Workbenches . . . 19
- 2.3 Architecture Description Languages . . . 21
- 2.4 The MontiArc Architecture Description Language . . . 23

3 Scope and Methodology **29**
- 3.1 Scenario . . . 30
- 3.2 Requirements . . . 33
  - 3.2.1 Modeling Requirements . . . 34
  - 3.2.2 Model Transformation Requirements . . . 36
- 3.3 Methodical Guidance . . . 37
  - 3.3.1 Extension with Behavior Languages . . . 41
  - 3.3.2 Architecture Modeling . . . 41
  - 3.3.3 Composed Code Synthesis . . . 43

4 C&C Architectures with Application-Specific Behavior **45**
- 4.1 MontiArcAutomaton ADL . . . 46
  - 4.1.1 Language Elements . . . 47
  - 4.1.2 Symbol Table . . . 51
  - 4.1.3 MontiArcAutomaton Symbol Table . . . 51
  - 4.1.4 Context Conditions . . . 53
  - 4.1.5 Transformations on the MontiArcAutomaton ADL AST . . . 65

4.2 Embedding Component Behavior Languages . . . 68
4.2.1 Syntactic Behavior Language Embedding . . . 69
4.2.2 Symbolic Language Integration . . . 73
4.2.3 Language Integration Infrastructure . . . 76
4.2.4 Language Integration Semantics . . . 81
4.3 Discussion . . . 82
4.4 Related Modeling Languages . . . 83

5 A Behavior Language with I/O$^{\omega}$ Automata. 87
5.1 Language Elements . . . 89
5.1.1 Automaton Declaration . . . 90
5.1.2 Inputs, Outputs, and Local Variables . . . 91
5.1.3 Values . . . 91
5.1.4 State Declarations, Initial States, and Initial Outputs . . . 92
5.1.5 Transitions . . . 93
5.1.6 Alternative Stimuli . . . 94
5.2 Symbol Table . . . 95
5.3 Context Conditions . . . 97
5.3.1 Uniqueness Conditions . . . 97
5.3.2 Convention Conditions . . . 98
5.3.3 Referential Integrity Conditions . . . 102
5.3.4 Type Correctness Conditions . . . 104
5.4 A Transformation on the Automata AST . . . 107
5.5 Integrating Automata into MontiArcAutomaton . . . 107
5.5.1 Semantics of Integrated Automata Models . . . 109
5.5.2 Integration Infrastructure . . . 109
5.6 Discussion . . . 110

6 Reusable Architectures through Bindings and Libraries 111
6.1 Modeling Platform-Independent Architectures . . . 113
6.2 Interface Libraries and Implementation Libraries . . . 118
6.2.1 BumperBot Interface Library . . . 121
6.2.2 JavaNXT Implementation Library . . . 123
6.2.3 Python ROS Implementation Library . . . 125
6.3 Deriving Platform-Specific Architectures . . . 128
6.4 Discussion and Related Approaches . . . 130

7 Compositional Code Generation 135
7.1 Code Generator Kinds . . . 137
7.2 Code Generator Description Language . . . 141
7.2.1 Language Elements . . . 141

7.2.2 Symbol Table . . . 146
7.2.3 Context Conditions . . . 147
7.3 Code Generator Composition . . . 152
7.3.1 Developing MontiArcAutomaton Generators . . . 154
7.3.2 Instantiating and Executing Composable Generators . . . 156
7.4 Two Compositional Code Generator Families . . . 159
7.4.1 A Code Generator Family for Java Systems . . . 160
7.4.2 A Code Generator Family for ROS Python Systems . . . 166
7.5 Discussion and Related Work . . . 173

8 Describing Component & Connector Applications 175
8.1 Application Configuration Language . . . 176
8.1.1 Language Elements . . . 176
8.1.2 Symbol Table . . . 179
8.1.3 Context Conditions . . . 180
8.2 Processing MontiArcAutomaton Applications . . . 191
8.3 Modeling MontiArcAutomaton Applications . . . 193
8.4 Discussion and Related Work . . . 196

9 Experiments 197
9.1 Evaluations . . . 197
9.1.1 NXT Java Coffee Delivery . . . 198
9.1.2 Robotino ROS Python Transport Services . . . 200
9.1.3 Robotino SmartSoft Java Transport Services . . . 205
9.2 Case Studies . . . 208
9.2.1 Lego NXT Distributed Toast Service . . . 208
9.2.2 Multi-Platform BumperBot . . . 209
9.2.3 The iserveU Hospital Logistics Project . . . 212
9.3 Discussion . . . 214

10 Conclusions and Future Work 217
10.1 Contributions . . . 217
10.2 Potential for Future Research . . . 219
10.3 Conclusion . . . 220

Bibliography 221

A Modeling Language Grammars 249
A.1 MontiArcAutomaton ADL Grammars . . . 249
A.1.1 MontiArc Grammar for Human Comprehension . . . 249
A.1.2 MontiArcAutomaton ADL Grammar for Human Comprehension . 250

A.1.3 MontiArcAutomaton ADL Grammar for MontiCore 251
A.2 Automata Grammars 254
A.2.1 Automata Grammar for Human Comprehension 254
A.2.2 Automata Grammar for MontiCore 255
A.3 Generator Description Grammar 255
A.4 Application Configuration Grammar 259

B Survey Materials **261**
B.1 NXT Java Coffee Delivery 261
B.2 Robotino ROS Python Transport Services 263
B.3 Robotino SmartSoft Java Transport Services 268

C Kinds of Names in MontiArcAutomaton **275**

D Diagram and Listing Tags **277**

E Curriculum Vitae **279**

List of Figures **281**

List of Listings **285**