

Katrin Hölldobler

MontiTrans: Agile, modellgetriebene
Entwicklung von und mit domänen-
spezifischen, kompositionalen
Transformationssprachen



Aachener Informatik-Berichte,
Software Engineering

Hrsg: Prof. Dr. rer. nat. Bernhard Rumpe

Band 36

**MontiTrans:
Agile, modellgetriebene Entwicklung von und mit
domänenspezifischen, kompositionalen
Transformations Sprachen**

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

Katrin Hölldobler,
M.Sc.RWTH
aus Haan

Berichter: Universitätsprofessor Dr. Bernhard Rumpe
Universitätsprofessor Dr. Ralf Lämmel

Tag der mündlichen Prüfung: 11. September 2018

Aachener Informatik-Berichte, Software Engineering

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

Band 36

Katrin Hölldobler
RWTH Aachen University

**MontiTrans: Agile, modellgetriebene
Entwicklung von und mit domänenspezifischen,
kompositionalen Transformationssprachen**

Shaker Verlag
Aachen 2018

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Zugl.: D 82 (Diss. RWTH Aachen University, 2018)

Copyright Shaker Verlag 2018

Alle Rechte, auch das des auszugsweisen Nachdruckes, der auszugsweisen oder vollständigen Wiedergabe, der Speicherung in Datenverarbeitungsanlagen und der Übersetzung, vorbehalten.

Printed in Germany.

ISBN 978-3-8440-6322-6

ISSN 1869-9170

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen

Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9

Internet: www.shaker.de • E-Mail: info@shaker.de

Kurzfassung

Modelle sind die zentralen Entwicklungsartefakte der modellgetriebenen Softwareentwicklung [CE00] und müssen entsprechend überarbeitet, weiterentwickelt und gewartet werden. Daher sind Modelltransformationen ebenfalls essenziell für die modellgetriebene Softwareentwicklung [SK03]. Während domänenspezifische Sprachen (DSLs) zur Modellierung mittlerweile weitverbreitet sind, sind spezifische Transformationssprachen rar. Stattdessen werden General Purpose Transformationssprachen (GPTLs) verwendet, die Transformationen basierend auf der internen Repräsentation der Modelle formulieren. Diese interne Repräsentation ist Modellierern und Domänenexperten in der Regel unbekannt, was deren Einbindung in den Entwicklungsprozess deutlich erschwert. Domänenspezifische Transformationssprachen (DSTLs) basieren auf der den Modellierern bekannten konkreten Syntax der DSL [BW07], wodurch sie spezifisch für die zugehörige DSL sind. Dies verringert den initialen Aufwand zum Erlernen der Transformationssprache, da der größte Teil der Syntax bereits bekannt ist und zusätzlich nur die Syntax der Transformationsoperatoren erlernt werden muss. Andererseits haben DSTLs durch ihre Zugehörigkeit zu einer DSL den Nachteil, dass für jede neu entwickelte DSL gleichzeitig oder für existierende DSLs nachträglich eine DSTL entwickelt werden muss. Aus diesem Grund erhöht sich der Aufwand der DSL-Entwicklung deutlich. Darüber hinaus erhöht sich der Aufwand weiter, wenn DSTLs zur Übersetzung zwischen verschiedenen Sprachen benötigt werden und entwickelt werden müssen. Zur Reduzierung dieses Aufwands wurde in [Wei12] ein erster DSTL-Generator vorgestellt. Dieser wurde mithilfe der Language Workbench MontiCore 2 entwickelt. Besondere Merkmale von MontiCore 4 sind die Möglichkeiten zur modularen Sprachdefinition durch Sprachkomposition [HLMSN⁺15b] sowie die an die Objektorientierung angelehnten Konzepte der Nichtterminalerweiterung als Erweiterung des Alternativenkonzepts von Grammatiken [Kra10]. Der bisherige DSTL-Generator ermöglicht eine Generierung von DSTLs aus MontiCore-Grammatiken und unterstützt ausschließlich monolithische Sprachdefinitionen. Die Mehrzahl der mit MontiCore entwickelten Modellierungssprachen sind hingegen kompositional definiert. Die bisher generierten DSTLs unterstützen ausschließlich Transformationen von Modellen innerhalb einer Modellierungssprache. Übersetzungen zwischen verschiedenen Sprachen oder die Migration von Modellen zwischen Sprachversionen ist somit nicht möglich. Des Weiteren wird bisher nur ein Teil der möglichen Konzepte von MontiCore-Grammatiken unterstützt, auf den sich die restlichen Konzepte abbilden lassen. Dadurch sind Sprachentwickler in der Sprachdefinition eingeschränkt.

Im Rahmen dieser Dissertation wird die generative Entwicklung und Verwendung von DSTLs in der modellgetriebenen Softwareentwicklung durch MontiTrans unterstützt. MontiTrans ermöglicht die Entwicklung neuer DSTLs und der zugehörigen Infrastruktur zur Spezifikation und Ausführung von Modelltransformationen. Für die Entwicklung von MontiTrans wurden die zuvor beschriebenen offenen Punkte aufgegriffen und die Generierung von DSTLs basierend auf den Ergebnissen aus [Wei12] weiterentwickelt.

Die wichtigsten Ergebnisse dieser Arbeit sind:

- Eine Systematik zur Ableitung von domänenspezifischen Transformationssprachen aus domänenspezifischen Sprachen.
- Ein Generator für domänenspezifische Transformationssprachen, der diese Systematik umsetzt, und alle Features des MontiCore 4-Grammatikformats unterstützt.
- Speziell angepasste DSTLs für die Sprachen CD4Analysis und CD4Code zur Klassendiagrammodellierung und die Architekturbeschreibungssprache MontiArc.
- Bibliotheken von wiederverwendbaren Transformationen für Klassendiagramme und Modelle der Architekturbeschreibungssprache MontiArc.
- Methodiken zum Einsatz von MontiTrans, DSTLs und Transformationen in der modellgetriebenen Softwareentwicklung.
- Eine Methodik zur handgeschriebenen Erweiterung der generierten DSTLs.

MontiTrans wurde sowohl zur Entwicklung der DSTLs als auch zur Entwicklung der Bibliotheken von wiederverwendbaren Transformationen verwendet. Hierdurch konnte gezeigt werden, dass MontiTrans ein umfassendes Werkzeug zur Entwicklung von DSTLs sowie für die Entwicklung von Modelltransformationen innerhalb modellgetriebener Softwareentwicklungsprojekte ist. MontiTrans erleichtert sowohl Sprachentwicklern die Entwicklung neuer DSLs und zugehöriger DSTLs als auch Transformationsentwicklern die Definition und Anwendung neuer Transformationen.

Abstract

Models are the central development artifact in model-driven software development [CE00]. These models need to be refactored, evolved and maintained. Thus, model transformations are indispensable for model-driven software development [SK03]. While using domain-specific languages (DSLs) has become common practice, tailored transformation languages are still uncommon. Instead, general-purpose transformation languages (GPTLs) are used that specify transformation based on the internal representation of models. Modelers and domain experts are typically unfamiliar with this representation, which hampers their integration into the development process. In contrast, DSTLs are based on the concrete syntax of the modeling language that is known by modelers [BW07]. Therefore DSTLs are specific for their corresponding modeling language. This reduces the initial effort to learn using the transformation language as the majority of the syntax is already familiar and only the transformation operators need to be understood in addition. However, a downside of DSTLs is that one DSTL needs to be developed for every DSL which results in significantly increased development effort for DSLs. In addition, this effort further increases in case DSTLs to translate between different modeling languages are needed. In [Wei12] a DSTL generator was developed to reduce this effort. This generator was developed using the language workbench MontiCore in version 2. MontiCore in version 4 provides modular language definitions via language composition [HLMSN⁺15b] as well as nonterminal extension features inspired by object orientation [Kra10]. The previous DSTL generator generates DSTLs from MontiCore grammars but only supports monolithic language definitions. However, the majority of languages developed using MontiCore is defined compositionally. Furthermore, the generated DSTLs only supported transformations of models within one modeling language. Thus, translation or migration between two languages was not possible. In addition, only a subset of the MontiCore grammar features were supported by the DSTL generator, which requires language developer to only use the supported features.

In this dissertation, MontiTrans supports the generative development of DSTLs and their usage in model-driven software development projects. MontiTrans facilitates developing new DSTLs and their infrastructure to specify and apply model transformations. For developing MontiTrans the open issues described above were considered and the generation of DSTLs based on the results of [Wei12] improved.

The main contributions of this thesis are:

- A systematic to derive domain-specific transformation languages from domain-specific languages.
- A generator for domain-specific transformation languages that implements this systematic and supports all features of the MontiCore grammar format.
- Specially tailored DSTLs for the modeling languages CD4Analysis and CD4Code to model class diagrams and the architecture description language MontiArc.
- Libraries of reusable transformations for class diagrams and MontiArc models.

- Methodologies to use MontiTrans, DSTLs and transformations in model-driven software development.
- A methodology to manually adapt and extend generated DSTLs.

MontiTrans was used to develop the DSTLs as well as the libraries of reusable transformations, which demonstrates that MontiTrans is a comprehensive tool to develop DSTLs and transformations within model-driven software development projects. MontiTrans facilitates developing DSLs and corresponding DSTLs for language engineers and specifying and applying transformations for transformation developers.

Danksagung

Ich möchte diese Worte nutzen, um mich bei den lieben Menschen zu bedanken, die mich auf dem Weg zur Promotion begleitet, unterstützt sowie motiviert oder an passender Stelle abgelenkt haben. Sie alle haben zum Gelingen dieses Vorhaben beigetragen.

Mein erster Dank gilt meinem Doktorvater Prof. Dr. Bernhard Rumpel für die Möglichkeit der Promotion am Lehrstuhl für Software Engineering. Durch konstruktive Diskussionen, Tipps und Ideen hat er diese Arbeit mitgeformt. Neben der spannenden, akademischen Arbeit durfte ich auch praktische Erfahrungen in verschiedenen Modellierungsprojekten sowie durch die Leitung der Arbeitsgruppe *Modellierung* und des MontiCore-Projekts sammeln. Auch für diese Erfahrungen bin ich dankbar.

Als nächstes möchte ich Prof. Dr. Ralf Lämmel für die Bereitschaft, das Zweitgutachten dieser Arbeit zu übernehmen, danken. Darüber hinaus bedanke ich mich bei Prof. Dr. Martin Grohe für die Leitung meines Prüfungskomitees und Prof. Dr. Thomas Noll für die Mitarbeit in diesem Komitee.

Bedanken möchte ich mich auch bei Dr. Ingo Weisemöller, der mich schon während meines Studiums für das Thema Modelltransformationen begeistern konnte und auf dessen Arbeit ich aufbauen konnte. Herzlich bedanken möchte ich mich auch bei allen Kollegen, die mich während und teils auch außerhalb meiner Zeit am Lehrstuhl begleitet haben: Kai Adam, Vincent Bertram, Marita Breuer, Lennart Bucher, Arvid Butting, Gereon Bürvenich, Manuela Dalibor, Imke Drave, Florian Donath, Robert Eikermann, Angelika Fleck, Sylvia Gunder, Timo Greifenberg, Dr. Arne Haber, Robert Heim, Lars Hermerschmidt, Dr. Christoph Herrmann, Gabriele Heuschen, Steffen Hillemacher, Andreas Horst, Oliver Kautz, Thomas Kurpick, Evgeny Kusmenko, Achim Lindt, Dr. Markus Look, Ben Mainz, Matthias Markthaler, Dr. Judith Michael, Dr. Pedram Mir Seyed Nazari, Dr. Klaus Müller, Antonio Navarro Pérez, Lukas Netz, Sebastian Oberhoff, Jerome Pfeiffer, Nina Pichler, Dr. Claas Pinkernell, Dr. Dimitri Plotnikov, Manuel Pützer, Deni Raco, Dr. Dirk Reiss, Dr. Jan Oliver Ringert, Dr. Alexander Roth, Dr. Martin Schindler, Steffi Schrader, Christoph Schulze, Brian Sinkovec, Galina Volkova, Max Voß, Michael von Wenckstern und Dr. Andreas Wortmann. Weiterhin möchte ich mich bei Achim, Andreas, Arvid, David, Dimitri, Heinz, Hille, Markus, Oliver, Robert, Steffen und Timo für das Sichten und Kommentieren früher Fassungen dieser Arbeit bedanken.

Auch möchte ich mich bei den Kollegen und Freunden bedanken, die immer ein offenes Ohr für mich hatten und mich mit Rat und Tat unterstützt haben. Darüber hinaus bedanke ich mich bei allen beteiligten Hiwis, Studenten und Abschlussarbeitern, die die Entwicklung von MontiTrans unterstützt haben. Vielen Dank für eure Unterstützung, sie hat mir sehr geholfen.

An dieser Stelle möchte ich außerdem meiner Familie und insbesondere meiner Mutter Ute meinen besonderen Dank aussprechen. Sie hat mir diesen Weg überhaupt erst ermöglicht, mich auf diesem Weg unterstützt und war jederzeit für mich da. Zusätzlich möchte ich mich auch bei Mechthild und Heinz, Hille und Detlef sowie Ralf bedanken. Auch auf ihre Unterstützung konnte ich mich immer verlassen.

Schließlich möchte ich mich noch ganz besonders bei meiner Partnerin Isa bedanken.

Sie hat mich stets unterstützt, mich motiviert und mir den Rücken freigehalten, sodass ich mich auf meine Promotion konzentrieren konnte, selbst wenn darunter die gemeinsame Zeit leiden musste. Ich danke dir!

Aachen, September 2018
Katrin Hölldobler

Inhaltsverzeichnis

I	Grundlagen	1
1	Einführung	3
1.1	Ziele dieser Arbeit	5
1.2	Wichtigste Ergebnisse der Arbeit	7
1.3	Aufbau der Arbeit	8
1.4	Verwandte Publikationen	11
2	Grundlagen der modellgetriebenen Softwareentwicklung	13
2.1	Die Language Workbench MontiCore	16
2.1.1	MontiCore-Grammatiken	17
2.1.2	Generierung der abstrakten Syntax	19
2.1.3	Codegenerierung mit MontiCore	20
2.1.4	Modulare Sprachdefinition und Sprachintegration mit MontiCore	22
2.2	Die Modellierungssprachen CD4Analysis und CD4Code	23
2.3	Die Modellierungssprache MontiArc	25
2.4	Grundlagen der Modelltransformation	26
2.5	Domänenspezifische Transformationssprachen in MontiCore	29
3	Rahmen und Anforderungen	31
3.1	Szenarien und Rollen	31
3.1.1	Rollen	31
3.1.2	Szenario 1: Neuentwicklung einer Transformationssprache	33
3.1.3	Szenario 2: Entwicklung von Transformationen	34
3.1.4	Szenario 3: Verwendung von Transformationen	36
3.2	Analyse von Transformationen	36
3.2.1	Analyse des Data Explorer Generator	38
3.2.2	Analyse des MontiCore-Generator	39
3.2.3	Analyse des MontiArc-Generator	40
3.2.4	Fazit der Analyse	41
3.3	Anforderungen an MontiTrans	42
3.3.1	Anforderungen an den DSTL-Generator	42
3.3.2	Anforderungen Transformationssprachen	46
3.3.3	Anforderungen Transformationen	48

II Domänenspezifische Transformationssprachen 49

4 Struktur und Operatoren der domänenspezifischen Transformationssprachen mit MontiTrans 51

4.1	Featureübersicht	53
4.2	Durchgängiges Beispiel	54
4.3	Pattern in konkreter Syntax	54
4.4	Schemavariablen	57
4.4.1	Schemavariablen für Namen	58
4.4.2	Abstraktion von Modellelementen durch Schemavariablen	60
4.4.3	Binden von (Teil-)Pattern an Schemavariablen	60
4.5	Modifikation	61
4.5.1	Ersetzen von Modellelementen	62
4.5.2	Hinzufügen von Modellelementen	63
4.5.3	Entfernen von Modellelementen	63
4.5.4	Verschieben von Modellelementen	64
4.5.5	Modifikation mittels Replacement Operator	65
4.6	Negative Elemente: Negative Application Condition	66
4.7	Collection Operator: Mengen von Elementen	68
4.8	Optional Operator: Pattern mit Optionalen Elementen	71
4.9	Folding Operator: Nichtisomorphes Matching	75
4.10	Zuweisung von Schemavariablen	77
4.11	Application Constraint	79
4.12	Direkte Manipulation und abschließende Aktionen	80
4.13	Built-Ins	83
4.13.1	Stringmanipulation	83
4.13.2	Listenbehandlung	84
4.13.3	Reporting and Logging	85
4.13.4	Templateerweiterung und Hook Points	86
4.14	Kontextbedingungen	86
4.14.1	Wohlgeformtheit bezüglich Schemavariablen	87
4.14.2	Wohlgeformtheit bezüglich der Kombination von Operatoren	92
4.14.3	Konventionen	96
4.14.4	Application Constraint, Zuweisungen und Direkte Manipulation	97
4.15	Diskussion und verwandte Arbeiten	97

5 Domänenspezifische Transformationssprachen für Klassendiagramme und MontiArc-Modelle 101

5.1	CDTrans: Eine DSTL für Klassendiagramme	102
5.1.1	CD4Code und CDTrans	103
5.1.2	CD4Code-spezifische Erweiterungen	111
5.2	MATrans: Eine DSTL für MontiArc-Modelle	115
5.2.1	MontiArc und MATrans	115
5.2.2	MontiArc-spezifische Elemente	124

5.3	MACDTrans: Eine DSTL für Klassendiagramme und MontiArc-Modelle	125
5.3.1	Von MontiArc zu Klassendiagrammen	126
5.3.2	Von Klassendiagrammen zu MontiArc	127
5.3.3	Koevolution von MontiArc-Modellen und Klassendiagrammen	128
5.3.4	Realisierung von MACDTrans	129
5.4	Aufbau von CD-, MA-, MACDTrans und der Basissprachen	130
5.5	Diskussion und verwandte Arbeiten	132
6	Ableitung und Generierung domänenspezifischer Transformationsprachen	135
6.1	Struktur von DSTLs für modular definierte Modellierungssprachen	136
6.2	Ableitungsregeln	137
6.2.1	Regel 1: Grammatikgrundgerüst	138
6.2.2	Regel 2: Interface-Nichtterminale und Externe Nichtterminale	139
6.2.3	Regel 3: Pattern und Schemavariablen	140
6.2.4	Regel 4: Replacement Operator	143
6.2.5	Regel 5: Negative Elemente	145
6.2.6	Regel 6: Collection Operator	146
6.2.7	Regel 7: Optional Operator	147
6.2.8	Regel 8: Startsymbol der DSTLs	148
6.3	Basissprache TFCCommons	149
6.4	Generierung einer neuen DSTL mit MontiTrans	149
6.5	Diskussion und verwandte Arbeiten	155
III	Domänenspezifische Transformationen	159
7	Wiederverwendbare Transformationen	161
7.1	Transformationsbibliothek für Klassendiagramme	162
7.1.1	Pull Up Attributes	163
7.1.2	Push Down Attribute	164
7.1.3	Pull Up Method	166
7.1.4	Push Down Method	167
7.1.5	Extract Super Class	169
7.1.6	Collapse Hierarchy	171
7.1.7	Extract Intermediate Class	172
7.1.8	Extract Class	173
7.1.9	Inline Class	175
7.1.10	Replace Delegation By Inheritance	176
7.1.11	Replace Inheritance By Delegation	177
7.1.12	Encapsulate Attribute	178
7.1.13	Replace Attribute by Association	179
7.1.14	Basistransformationen	180
7.2	Transformationsbibliothek für MontiArc	181
7.2.1	Normalisierungen zur Codegenerierung	181

7.2.2	Wrapper für Porttypen	189
7.2.3	Vereinfachung von MontiArc-Modellen	190
7.2.4	Eliminierung generischer Komponenten	191
7.3	Diskussion und verwandte Arbeiten	193
8	Generierung der Transformationen	195
8.1	Aufbau des Generators und Ablauf der Generierung	197
8.2	Transformationsregeln in ODRules-Notation	199
8.3	Pattern Matching	204
8.3.1	Suchplangesteuertes Pattern Matching	205
8.3.2	Modularisierung und Überprüfung des Application Constraints	208
8.3.3	Pattern Matching für den Optional und den Collection Operator	212
8.4	Diskussion und verwandte Arbeiten	215
9	MontiTrans-basierte Methodiken	219
9.1	Entwicklung einer neuen DSTL	220
9.1.1	Methodik	220
9.1.2	Konfiguration und Ausführung des DSTL-Generators	222
9.2	Methodik zur Erweiterung der generierten DSTLs	226
9.3	Entwicklung von Transformationsregeln	228
9.3.1	Methodik	228
9.3.2	Konfiguration und Ausführung des Java-Generators	231
9.4	Methodik zur Verwendung und Entwicklung komplexer Transformationen in Java	235
9.4.1	Anwendung und Kombination von Transformationsregeln	236
9.4.2	Verwendung von Parametern	237
9.4.3	Verwendung des Templateerweiterungsmechanismus	238
9.5	Zusammenfassung	239
IV	Epilog	241
10	Zusammenfassung und Ausblick	243
	Literaturverzeichnis	249
V	Anhänge	273
A	Markierungen in Abbildungen und Listings	275
B	Abkürzungen	277
C	Modelle und Grammatiken	279
C.1	Grammatik der DSLs CD4Analysis und CD4Code	279

C.2	Grammatik der DSTL CDTrans	281
C.3	Grammatik der DSTL MATrans	295
C.4	Abgeleitete Automaton DSTL	305
C.5	DSTL-Generierung mit MontiTrans	307
C.6	Generierung von Java-Implementierungen für Transformationsregeln . . .	308
D	Curriculum Vitae	309
	Abbildungsverzeichnis	311
	Listings	315
	Tabellenverzeichnis	319