Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P

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
Martin Schindler

aus Dortmund

Berichter: Universitätsprofessor Dr. rer. nat. Bernhard Rumpe
Universitätsprofessor Dr. rer. nat. Albert Zündorf

Tag der mündlichen Prüfung: 23. Dezember 2011

Aachener Informatik-Berichte, Software Engineering

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

Band 11

Martin Schindler

Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P

Shaker Verlag
Aachen 2012

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, 2011)

Printed in Germany.

ISBN 978-3-8440-0864-7
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

"It's not the strongest who survive, nor the most intelligent, but the ones most adaptable to change."

Charles Darwin

Kurzfassung

Einer der vielversprechendsten Ansätze, um der steigenden Komplexität von Softwareentwicklungsprojekten zu begegnen, ist die Abstraktion durch Modellierung, für die sich in den letzten Jahren die UML etabliert hat. Diese eher schwergewichtige und in Teilen redundante Familie von Modellierungssprachen wurde mit der UML/P auf wesentliche Kernnotationen reduziert und hinsichtlich der Anforderungen agiler, modellgetriebener Entwicklungsprozesse optimiert. In Kombination mit Java als Aktionssprache erlaubt die UML/P eine vollständige Modellierung und Qualitätssicherung von Softwaresystemen, deren technische Realisierung mittels Generatoren aus den Modellen abgeleitet wird.

Die praktische Anwendung der agilen Konzepte und Modellierungstechniken der UML/P wird mit dieser Arbeit erstmalig durch eine Werkzeuginfrastruktur unterstützt. Zusammengefasst sind die wichtigsten Ergebnisse wie folgt:

- Die *Modellierung* von Struktur, Verhalten und Qualitätssicherung von Softwaresystemen wird durch eine an Java angelehnte textuelle Notation der UML/P unterstützt. Diese bietet eine kompakte, effiziente und werkzeugunabhängige Modellerstellung, eine Strukturierung der Modelle in Pakete, eine Unterstützung von Sprachprofilen sowie explizite Modellbeziehungen durch qualifizierte Namensreferenzen und Importe.
- Die *Kontextanalyse* erlaubt eine modell- und sprachübergreifende Konsistenzsicherung der Modelle und kann flexibel an Sprachprofile oder unterschiedliche Modellabstraktionen angepasst und erweitert werden. Dabei wird insbesondere Java als abstrakte Aktionssprache auf Modellebene angehoben und interpretiert.
- Die *Codegenerierung* wird mit Templates als zentrales Artefakt für Entwicklung, Konfiguration und Ausführung von Generatoren modular bezüglich Sprachen und Modellen umgesetzt. Das Tracing von Generat und dessen Quellen, die Optimierung auf lesbare Templates, die automatisierte Ableitung des Datenmodells und die Anbindung von Java vereinfacht die Anpassung und Erweiterung der Generierung im Projektkontext. Dazu wird eine Methodik zur schrittweisen Ableitung von Generatoren aus exemplarischem Quellcode definiert. Konzepte für die Generierung von Produktiv- und Testcode aus UML/P-Modellen sowie ein Verfahren zur sprachunabhängigen Messung der Testüberdeckung auf Modellen ergänzen diesen Ansatz und demonstrieren, wie aus abstrakten Modellen eine kompositionale, erweiterbare und testbare Architektur für das Zielsystem abgeleitet werden kann.
- Die *Werkzeuginfrastruktur* wird als modulares und leichtgewichtiges Framework umgesetzt. Dabei wird jede Sprache als eigenständige Komponente realisiert, die jeweils die grammatikbasierte Sprachdefinition, Sprachverarbeitung, Kontextbedingungen und Generatoren unabhängig von anderen Sprachen enthält. Dies erlaubt eine flexible Erweiterung und variable Zusammensetzung der Sprachen der UML/P nach dem Baukastenprinzip.

Insgesamt ist damit eine umfassende Werkzeuginfrastruktur zur agilen modellgetriebenen Entwicklung mit der UML/P entstanden, die sich flexibel erweitern und an Technologien sowie Domänen anpassen lässt.

Abstract

One of the most promising approaches for handling the increasing complexity of software development projects is the abstraction using modeling for which the UML has been established during the last years. With the UML/P this rather heavyweight and partly redundant family of modeling languages got reduced to essential core notations and optimized according to the requirements of agile, model-driven development processes. Adding Java as action language, the UML/P allows entire modeling and quality assurance of software systems whereas the technical realization is derived from the models using generators.

With this work the practical application of the agile concepts and modeling techniques of the UML/P are supported by a tooling infrastructure for the first time. Summarized the main results are as follows:

- The *modeling* of structure, behavior, and quality assurance of software systems is supported by a Java-like textual notation for the UML/P. This offers a compact, efficient, and tooling-independent model creation, a structuring of models in packages, a support of language profiles as well as explicit model relations using qualified name references and imports.
- The *context analysis* allows model- and language-crossing consistency checks of the models and is flexibly adaptable and extendable for language profiles or different model abstractions. Particularly the context analysis interprets Java against the models so that Java is raised to the model-level as abstract action language.
- The *code generation* is realized using templates as central artifact for development, configuration, and execution of generators in a modular manner concerning languages and models. Tracing of generated code and its sources, optimizations for readable templates, automatic derivation of the data model, and invoking of Java code simplifies the adaption and extension of the generators within a development project. In addition, a method for a stepwise development of generators from exemplary source code is defined. The approach is complemented by concepts for production and test code generation from UML/P-models as well as by a technique for a language-independent calculation of test coverage on models. This demonstrates how a compositional, extendable, and testable architecture can be derived from models for the target system.
- The *tooling infrastructure* is implemented as a modular and lightweight framework. Each language is realized as a self-contained component, which combines the grammar-based language definition, language processing, context conditions and generators independently from other languages. This allows a flexible extension and variable composition of the UML/P languages in a modular manner.

To conclude, this work offers a comprehensive tooling infrastructure for the agile model-driven development with the UML/P which is flexibly extendable and adoptable to technology as well as to domains.

Danksagung

Während meiner Promotion haben mich viele liebe Menschen begleitet und unterstützt und so zum Erfolg dieser Dissertation beigetragen. Dafür möchte ich mich hier herzlich bedanken.

Ganz besonderer Dank gebührt meinem Doktorvater Prof. Dr. Bernhard Rumpe, auf dessen Habilitation diese Arbeit aufbaut. Die gemeinsamen Diskussionen, seine konstruktiven und weitsichtigen Anregungen sowie seine Erfahrung haben maßgeblich zu dieser Arbeit beigetragen. Darüber hinaus habe ich im Rahmen meiner Lehrtätigkeit, der Betreuung des Journals "Software and Systems Modeling" (SoSyM) sowie den zahlreichen Projekten im Industrie- und Forschungsumfeld in dieser Zeit viel von ihm über praxisrelevante Techniken und Methoden des Software Engineerings gelernt. Für diese vielschichtige Erfahrung und das sich daraus ergebende abwechslungsreiche und spannende Arbeitsumfeld möchte ich ihm ebenfalls danken.

Prof. Dr. Albert Zündorf danke ich herzlich für sein Interesse an meiner Arbeit und die Übernahme des Zweitgutachtens. Eine ganz besondere Herausforderung war die Organisation der Promotionsprüfung einen Tag vor Weihnachten. Möglich wurde dies erst durch den Einsatz und der Bereitschaft von Prof. Dr. Dr.h.c. Wolfgang Thomas und Prof. Dr. Stefan Kowalewski, selbst an diesem Tag noch die Prüfung durchzuführen.

Für das hervorragende Arbeitsklima, die intensiven Diskussionen und Anregungen, den Ideenaustausch, die Zusammenarbeit in gemeinsamen Projekten und Papieren sowie für die vielen gemeinsamen Erlebnisse möchte ich meinen Kollegen des Instituts für Software Systems Engineering der TU Braunschweig und des Lehrstuhls für Software Engineering der RWTH Aachen danken. Hervorzuheben sind hier insbesondere Dr. Hans Grönniger, Arne Haber, Dr. Holger Krahn, Claas Pinkernell, Dr. Steven Völkel und Ingo Weisemöller, deren Arbeiten rund um MontiCore, Sprachentwicklung und -semantik zu gegenseitigen Anregungen, Impulsen und Synergieeffekten geführt und so eine gemeinsame Forschung sowie kooperatives Arbeiten möglich gemacht haben. Inspiriert wurde ich dabei immer wieder von Holgers fachlicher Kompetenz und Stevens pragmatischer Herangehensweise. In der Endphase meiner Implementierung haben mich darüber hinaus Marita Breuer und Galina Volkova bei der Umsetzung der Symboltabellen und Kontextbedingungen besonders unterstützt. Danken möchte ich auch Ingo, Steven, Hans und meiner Frau Daniela für die kritische Durchsicht meiner Ausarbeitung, sowie Claas Oppitz für die Gestaltung der TripleLogo-Tiere. Auch wenn namentlich nicht genannt, so haben doch alle meine Kollegen dazu beigetragen, dass mir meine Dissertation als eine spannende und schöne Zeit in Erinnerung bleiben wird. Dabei sorgten u.a. gemeinsame Kickerturniere für den sportlichen Ausgleich, bei denen mir insbesondere Steven regelmäßig meine Grenzen aufgezeigt hat.

Nicht zuletzt gilt mein Dank meiner Familie und Freunden für ihre Unterstützung und ihr Verständnis, wenn ich mich wieder mal nicht von meiner Arbeit losreißen konnte. Meinen Eltern möchte ich darüber hinaus für ihre stete Unterstützung, den Rückhalt und das Vertrauen in meinen Lebensweg danken. Ebenfalls danke ich meinen Schwiegereltern, dass auch sie immer für mich da sind. Mein besonderer Dank gebührt schließlich meiner Frau Daniela für ihre uneingeschränkte Unterstützung, Liebe und die unglaubliche Geduld, ohne die diese Arbeit nicht möglich gewesen wäre.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation und Kontext . . . 2
1.2 Forschungsfragen, -ziele und -herausforderungen . . . 4
1.3 Aufbau der Arbeit . . . 6

2 Grundlagen modellbasierter Softwareentwicklung **9**
2.1 Grundlegende Begriffe und Ziele . . . 10
2.2 Existierende Konzepte und Werkzeuge . . . 13
2.3 Entwicklung von Sprachen und Werkzeugen . . . 17

3 Die Sprache UML/P **21**
3.1 Klassendiagramme . . . 22
3.2 Objektdiagramme . . . 32
3.3 Statecharts . . . 35
3.4 Sequenzdiagramme . . . 43
3.5 Testspezifikationssprache . . . 50
3.6 OCL/P . . . 53
3.7 Java/P . . . 57
3.8 Gemeinsame Betrachtung der Teilsprachen . . . 60
3.9 Unterschiede zur ersten Fassung der UML/P . . . 66
3.9.1 Allgemeine Notation und neue Konzepte . . . 67
3.9.2 Semantik und Spracheinbettung . . . 69
3.9.3 Lokale Unterschiede . . . 71

4 Kontextbedingungen **75**
4.1 Grundlagen . . . 75
4.2 Sprachinterne Intra-Modell-Bedingungen . . . 80
4.2.1 Allgemeine Bedingungen . . . 80
4.2.2 Klassendiagramme . . . 83
4.2.3 Objektdiagramme . . . 91
4.2.4 Statecharts . . . 94
4.2.5 Sequenzdiagramme . . . 97

4.2.6 Testspezifikationssprache . . . 99
4.3 Sprachinterne Inter-Modell-Bedingungen . . . 99
4.3.1 Klassendiagramme . . . 102
4.3.2 Objektdiagramme . . . 102
4.4 Sprachübergreifende Intra-Modell-Bedingungen . . . 102
4.4.1 Allgemeine Bedingungen . . . 103
4.4.2 Klassendiagramme . . . 103
4.4.3 Objektdiagramme . . . 105
4.4.4 Statecharts . . . 106
4.4.5 Sequenzdiagramme . . . 106
4.5 Sprachübergreifende Inter-Modell-Bedingungen . . . 107
4.5.1 Klassendiagramme . . . 107
4.5.2 Objektdiagramme . . . 107
4.5.3 Statecharts . . . 110
4.5.4 Sequenzdiagramme . . . 112
4.5.5 Testspezifikationssprache . . . 114

5 Templatebasierte Codegenerierung 117
5.1 Grundlagen . . . 119
5.1.1 Modelltransformationen . . . 119
5.1.2 MontiCore . . . 121
5.1.3 FreeMarker . . . 123
5.2 Aufbau eines Generators . . . 131
5.3 Vorgehensweise bei der Implementierung eines Generators . . . 142
5.4 Modifikation und Erweiterung eines Generators . . . 146
5.5 Komposition von Generatoren . . . 148

6 Konzepte der Codegenerierung 153
6.1 Umgang mit Unterspezifikation . . . 155
6.2 Projektspezifische Variabilität . . . 159
6.3 Generierung von Produktivcode . . . 162
6.3.1 Delegator-basierte Komposition mit Factories . . . 164
6.3.2 Delegator-basierte Komposition per Namenskonvention . . . 168
6.3.3 Aspektorientierte Komposition . . . 170
6.4 Codeinstrumentierung . . . 173
6.5 Generierung von Testcode . . . 175
6.6 Testüberdeckung und -güte auf Modellen . . . 180

7 Das UML/P-Framework 189
7.1 Projektstruktur . . . 193
7.2 Symboltabellen . . . 202

7.3 Framework für Kontextbedingungen . . . 209
7.4 Generierungsframework . . . 214
7.5 Refactoringframework . . . 217
7.6 Nutzung des UML/P-Frameworks . . . 220

8 Diskussion und verwandte Arbeiten 229
8.1 Sprache und Modellierungskonzepte . . . 229
8.2 Kontextbedingungen . . . 232
8.3 Generatoren und Templatesprachen . . . 233
8.4 Modellgetriebene Ansätze und Werkzeuge . . . 238

9 Zusammenfassung und Ausblick 241

A Abkürzungen 247

B Diagramm- und Quellcodemarkierungen 249

C Grammatiken der UML/P 251
C.1 Literale . . . 252
C.2 Typen . . . 257
C.3 Gemeinsame Sprachanteile der UML/P . . . 263
C.4 Klassendiagramme . . . 266
C.5 Objektdiagramme . . . 271
C.6 Statecharts . . . 273
C.7 Sequenzdiagramme . . . 277
C.8 Testspezifikationssprache . . . 280
C.9 OCL/P . . . 281
C.10 Java . . . 295

D Übersicht der Templates 315

E Lebenslauf 319

Abbildungsverzeichnis 323

Tabellenverzeichnis 325

Quellcodeverzeichnis 328

Literaturverzeichnis 347

Stichwortverzeichnis 349