

ON THE FORMALIZATION OF  
MODEL-DRIVEN SOFTWARE ENGINEERING

Vom Promotionsausschuss der  
Technischen Universität Hamburg-Harburg  
zur Erlangung des akademischen Grades

Doktor-Ingenieur

genehmigte Dissertation

von

Miguel Alfredo GARCIA GUTIERREZ

aus San Juan, Argentinien

2009

1. Gutachter: Prof. Dr. Ralf Möller, TU Hamburg-Harburg.
  2. Gutachter: Prof. Dr. Friedrich H. Vogt, TU Hamburg-Harburg.
- Tag der mündlichen Prüfung: 27. Februar 2009

---

## Abstract

Model-Driven Software Engineering (MDSE) encompasses traditional areas of language design, tool engineering, and system validation and verification, following a unified conceptual and technical framework (metamodeling, declarative model transformations, model-based analysis). This work presents design cases of methodologies and tools for MDSE, where the state-of-the-art is advanced as a result of applying formal techniques. The contributions encompass (a) the application of metamodeling techniques to industrially relevant languages, capturing their static semantics in a machine-processable manner; (b) the formulation of a methodology for the design-time certification of transformation algorithms; (c) the design of algorithms for efficiently evaluating Object Constraint Language (OCL) invariants for both the secondary-storage and main-memory cases; and (d) several contributions focused on the generation of Integrated Development Environments (IDEs) derived from language definitions for Domain Specific Languages (DSLs). Venues for further progress and an appraisal of the impact of our research are also reported.

## Kurzfassung

Modellgetriebene Softwareentwicklung (MGSE) umfasst die traditionellen Gebiete des Sprachentwurfs, der Werkzeugentwicklung, und der Systemvalidierung in einem einheitlichen konzeptuellen und technischen Rahmen (Meta-Modellierung, deklarative Modell-Transformation, und modellbasierte Analyse). Diese Arbeit untersucht Entwurfsvfälle für Methodologien und Werkzeuge im Rahmen der MGSE, wobei der Stand der Kunst durch die Anwendung formaler Techniken erhöht wird. Die Forschungsbeiträge dieser Arbeit umfassen (a) die Anwendung von Techniken der Meta-Modellierung auf industriell relevante Programmiersprachen, (b) die Formulierung einer methodischen Vorgehensweise zur Zertifizierung von Transformations-Algorithmen zur Entwurfszeit, (c) der Entwurf von Algorithmen zur effizienten Evaluierung von in der Object Constraint Language (OCL) formulierten Invarianten im Sekundär- und im Hauptspeicher und (d) diverse Beiträge zur Generierung von Integrierten Entwicklungsumgebungen (Integrated Development Environments, IDEs) aus Sprachspezifikationen für Domänenspezifische Sprachen (Domain Specific Languages). Ebenso wird ein Ausblick für zukünftige Forschungsarbeiten gegeben, und der Einfluss unserer Forschungsarbeiten auf industrielle Praxis diskutiert.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation for this Research . . . . .	1
1.2	Thesis Statement . . . . .	2
1.3	Background . . . . .	3
1.3.1	Design Patterns vs. Language Extensions vs. DSLs . . . . .	3
1.3.2	Logical Consistency of Data Models . . . . .	5
1.3.3	Logical Consistency of Behavioral Specifications . . . . .	5
1.3.4	Verification of OO Programs . . . . .	7
1.4	Contributions . . . . .	9
1.5	List of Publications . . . . .	10
1.6	Dissemination in Industry and Standardization Bodies . . . . .	12
1.7	Outline of the Dissertation . . . . .	13
1.8	Acknowledgements . . . . .	14
<b>2</b>	<b>The Purpose of Language Metamodeling</b>	<b>15</b>
2.1	Sample Shortcomings of the JPQL Spec . . . . .	18
2.2	Consistency and Completeness Enforced by Language Metamodeling	20
2.3	Selected Examples of Additional Corner Cases . . . . .	23
2.3.1	Visibility of Declarations . . . . .	23
2.3.2	Reduction of Datasets into Groups . . . . .	25
2.3.3	Path Expressions . . . . .	26
2.4	Static Semantics and Database Schema . . . . .	28
2.5	Related Work . . . . .	30
2.5.1	Metamodeling-based Approaches . . . . .	30
2.5.2	Grammar-based Approaches . . . . .	31

---

2.6	Summary . . . . .	31
<b>3</b>	<b>Metamodel-based Specification of Type Safety</b>	<b>37</b>
3.1	Type Systems . . . . .	39
3.2	Well-formed Types in EMF Generics . . . . .	41
3.2.1	Declaration of a Generic Type . . . . .	42
3.2.2	Type Invocation . . . . .	42
3.2.3	Subtyping Between Two Parameterized Types . . . . .	43
3.3	Related Work . . . . .	45
3.3.1	Typing of Object-Oriented Programs . . . . .	45
3.3.2	Improvements to the Java Type System . . . . .	46
3.4	Evaluation . . . . .	48
<b>4</b>	<b>Logical Consistency of Metamodels</b>	<b>51</b>
4.1	Application of Alloy Model Generation to Description Logics . . . . .	53
4.2	Translation from Description Logics . . . . .	55
4.2.1	Translation Rules for <i>ALC</i> . . . . .	56
4.2.2	Translation of <i>SHIQ</i> and <i>SROIQ</i> . . . . .	58
4.3	Case Studies . . . . .	59
4.3.1	Model Inspection by Counterexample Extraction . . . . .	59
4.3.2	Counterexamples for a Subsumption Assumption . . . . .	60
4.3.3	Counterexamples for a Concept Equivalence Assumption . . . . .	62
4.4	Evaluation of Practical Usefulness . . . . .	62
4.5	Outlook . . . . .	65
<b>5</b>	<b>Translation of OCL Specifications</b>	<b>67</b>
5.1	Target Software Architecture . . . . .	70
5.2	Processing of OCL Abstract Syntax Trees . . . . .	71
5.2.1	Basic API for Visitors . . . . .	73
5.2.2	Usage of Generics when Processing OCL ASTs . . . . .	75
5.2.3	Common Steps in Writing OCL Visitors . . . . .	76
5.2.4	Further Techniques to Process OCL ASTs . . . . .	79
5.3	Compilation Phases . . . . .	83

---

5.3.1	Information Initially Available to the Compiler . . . . .	83
5.3.2	Types Conversion . . . . .	84
5.3.3	Expressions Translation . . . . .	86
5.4	Translation Patterns . . . . .	90
5.4.1	Extending the Compiler . . . . .	91
5.5	Refactoring of OCL Expressions . . . . .	92
5.6	Performance . . . . .	94
5.7	Integration in an MDSE Toolchain . . . . .	95
5.8	Related Work and Evaluation . . . . .	96
<b>6</b>	<b>Automating the Embedding of Domain Specific Languages</b>	<b>101</b>
6.1	Embedded DSLs and Static Semantics . . . . .	102
6.2	Code Idioms in APIs for Embedded DSLs . . . . .	105
6.3	Checking Static Semantics During Editing . . . . .	107
6.4	Processing DSL Statements Beyond Checking of Static Semantics . .	109
6.4.1	Existing IDE Infrastructure for DSL Processing . . . . .	109
6.4.2	In-place Translation . . . . .	110
6.4.3	Statement-level Annotations . . . . .	112
6.4.4	DSL-specific Views . . . . .	113
6.5	Related Work . . . . .	113
6.5.1	DSL Embedding in Scala . . . . .	113
6.5.2	Static Analysis of XML Artifacts . . . . .	114
6.6	Summary . . . . .	115
<b>7</b>	<b>Generation of Authoring Environments from Language Specifications</b>	<b>117</b>
7.1	State of the Art in IDE Generation . . . . .	119
7.2	Generation of Parsing Infrastructure . . . . .	120
7.3	Functional Categories to Support by DSL Text Editors . . . . .	123
7.4	Case Study: Textual Notation for a Statechart Language . . . . .	124
7.4.1	Arguments In Favor of a Textual Notation . . . . .	125
7.4.2	An Example: Statechart of Telephone Object . . . . .	126
7.5	IDE support for OCL . . . . .	129
7.5.1	OCL Text Editor . . . . .	129

---

7.5.2	A Usability Feature in Focus: Mark Occurrences . . . . .	131
7.5.3	Candidate Extensions . . . . .	134
7.6	Related Work . . . . .	135
7.6.1	Dual Syntaxes . . . . .	136
7.6.2	Verbalization into Controlled Natural Language . . . . .	136
7.7	Outlook . . . . .	137
<b>8</b>	<b>Bidirectional Synchronization in Multiview IDEs</b>	<b>141</b>
8.1	Benefits of the Proposed Approach . . . . .	143
8.2	Candidate Approaches Distilled . . . . .	145
8.2.1	Program Inversion, Data Synchronization, and Virtual View Update . . . . .	146
8.2.2	Graph-grammars and QVT-Relations . . . . .	148
8.3	Integration in an EMOF-based Modeling Infrastructure . . . . .	150
8.3.1	Operators for Two-way Transformations: TwEcore . . . . .	151
8.3.2	Encoding of EMOF Models Using Inductive Data Types . . . . .	152
8.3.3	Diagrammatic Views and Geometric Constraint Solvers . . . . .	153
8.4	Related Work . . . . .	154
8.5	Evaluation . . . . .	156
<b>9</b>	<b>Design-time Certification of Transformation Algorithms</b>	<b>159</b>
9.1	Formalization of Essential MOF + OCL for Model Checking . . . . .	162
9.1.1	Translating EMOF into <sup>+</sup> CAL . . . . .	163
9.1.2	Translating OCL into <sup>+</sup> CAL . . . . .	164
9.2	Certification Process . . . . .	166
9.2.1	Directly Specifying Transformations in Terms of EMOF . . . . .	166
9.3	Certifying a Non-trivial In-place Transformation: Schorr-Waite . . . . .	167
9.3.1	A Graph-marking Algorithm for Garbage Collection . . . . .	167
9.3.2	Alloy Formalization of Operations . . . . .	171
9.3.3	Model Checking with <sup>+</sup> CAL . . . . .	172
9.4	Related Work: Alternative and Complementary Approaches . . . . .	172
9.5	Evaluation and Future Work . . . . .	174

---

<b>10 Model Transformation Based on Pattern Matching</b>	<b>179</b>
10.1 Term Rewriting . . . . .	180
10.2 Formalization Approach . . . . .	182
10.3 Static semantics, OCL Formulation . . . . .	184
10.3.1 WFRs for Templates . . . . .	186
10.3.2 WFRs Around <code>RelationCallExps</code> . . . . .	190
10.3.3 Interplay Between Relation Overriding and the Target Model .	191
10.3.4 Types of Members of a <code>CollectionTemplateExp</code> . . . . .	193
10.3.5 A Sidenote on Terminology . . . . .	195
10.4 Dynamic Semantics, Alloy Formulation . . . . .	195
10.4.1 Methodology . . . . .	197
10.4.2 Case Study . . . . .	199
10.5 Related Work . . . . .	201
10.6 Evaluation . . . . .	203
<b>11 Efficient Run-time Integrity Checking: Software Repositories</b>	<b>205</b>
11.1 Integrity Checking in Software Repositories . . . . .	207
11.2 Expressiveness and Runtime Cost of Integrity Checking . . . . .	208
11.3 Incremental Integrity Checking for OCL . . . . .	210
11.4 Computationally Complete Constraint Language . . . . .	212
11.5 Translation of OCL into Monoid Calculus . . . . .	214
11.5.1 The Monoid Calculus . . . . .	215
11.5.2 Translation Rules . . . . .	217
11.6 Optimizations with Monoid Calculus . . . . .	219
11.7 Related Work . . . . .	224
11.8 Evaluation . . . . .	226
<b>12 Efficient Run-time Integrity Checking: Main-memory</b>	<b>229</b>
12.1 Preconditions for Incrementalization . . . . .	231
12.1.1 Overlapping Subproblems . . . . .	231
12.1.2 Referential Transparency . . . . .	232
12.2 Incrementalization . . . . .	233
12.2.1 The DITTO Instrumentation Algorithm . . . . .	235



---

12.2.2	Design Considerations . . . . .	238
12.2.3	A First Example of OCL Incrementalization . . . . .	240
12.2.4	Termination Behavior of OCL Expressions . . . . .	243
12.3	Incrementalization: Compile-time Tasks . . . . .	243
12.3.1	DDG Lookup . . . . .	244
12.3.2	Implicit Arguments and Their Setters . . . . .	244
12.3.3	Operations on Collections and their Mutators . . . . .	246
12.3.4	Subcomputations . . . . .	247
12.4	Incrementalization: Runtime Tasks . . . . .	247
12.4.1	Update Phase of a Transaction . . . . .	247
12.4.2	Commit Phase Activities . . . . .	248
12.5	Consequences of the Design Choices Made . . . . .	250
12.6	Future Work: Shared-memory Transactions . . . . .	251
12.6.1	Motivation . . . . .	251
12.6.2	Runtime Detection of Data Races . . . . .	253
12.6.3	Initial Assessment . . . . .	254
12.7	Related Work . . . . .	255
12.8	Evaluation . . . . .	257
<b>13</b>	<b>Conclusions</b>	<b>259</b>
13.1	Evaluation of the Research Hypothesis . . . . .	259
13.2	Closing Remarks . . . . .	260