
List of Figures

2.1	Simple Bootstrapping Example: Getting a Native Compiler for Language L	13
2.2	Basic MD* terminology (by Stahl et al. [Sta+07, p. 28], translated into English)	15
2.3	Four-Level Example of MDA's Metamodel Hierarchy (based on [Obj10a, p. 19])	18
2.4	Using a template engine for code generation	30
3.1	The Extreme Model-Driven Development (XMDD) approach	40
3.2	The jABC user interface [JSM10]	44
3.3	Service adapter pattern for realizing a SIB's behavior	50
3.4	Hierarchical models in jABC	51
3.5	The LocalChecker plugin in jABC [JMS11]	55
3.6	Executing an SLG with the Tracer	62
3.7	User interface of the GEAR plugin	68
4.1	Genesys architecture and involved roles	75
4.2	Data type mapping infrastructure in Genesys	81
4.3	Specifying variants via hierarchical modeling	86
4.4	The DocuGenerator main model (topmost hierarchy level) ...	89
4.5	The Initialize Docu Generator model (second hierarchy level)	90
4.6	The Load Models model (third hierarchy level)	91
4.7	The Generate Documentation model (second hierarchy level)	91
4.8	The Generate Model Pages model (third hierarchy level) ...	92
4.9	The Generate Markup for SIBs model (fourth hierarchy level)	93
4.10	The model hierarchy of the Documentation Generator	94
4.11	Translating the SLGs of the Documentation Generator to Java code	95

4.12	Left hand side: Inspector for editing a code generator's metadata, Right hand side: Setting up a benchmark	97
4.13	Benchmark results visualized in tabular or bar chart form ...	98
4.14	Creating a configuration for using a code generator in jABC...	100
5.1	Generation concept of the Java Class Extruder	104
5.2	Java Class Extruder: Processing the input SLGs	106
5.3	Bootstrapping the Java Class Extruder by means of the Tracer	107
5.4	Generation concept of the Java Class Generator	112
5.5	Basic control flow patterns and their translation into code ...	116
5.6	The fork-join control flow pattern and its translation to code	116
5.7	An unstructured SLG and its equivalent structured pendant	117
5.8	Transformation of a fork-join construct to hierarchical submodels	118
5.9	Java Class Generator: Root Service Logic Graph (SLG) (1), Model transformations (2)	119
5.10	Java Class Generator: Detection of Control Flow Patterns ...	121
5.11	Lightweight data structure for SLGs (excerpt)	122
5.12	Generation concept of the Java Class Generator's "Interpreter Variant"	123
5.13	Transformation phase of the Genesys Code Generator Generator	125
5.14	Reuse of models among the Java Class Generator Variants ...	130
5.15	Distribution of Service Independent Building Block (SIB) bundles in the different Java code generators	132
5.16	Genealogical tree of code generators for jABC	137
6.1	Verification & Validation in Genesys	156
6.2	Example: Verifying the Java Class Extruder with GEAR and the FormulaBuilder	159
6.3	Constraints from the <code>actionOccurrence</code> category	161
6.4	Constraints from the <code>actionOrder</code> category	164
6.5	The new "Handle By" pattern with scope "before", meaning "Handle every A by P before Q".	166
6.6	A constraint checking for the complete handling of input SLGs using the "Handle By" pattern	167
6.7	Model-Driven Test Development (MDTD) in jABC	169
6.8	Strategy for testing the jABC code generators	171
6.9	Example SLGs modeling test inputs	173
6.10	The testing concept from Fig. 6.8, modeled in jABC	174
6.11	Excerpt from the test suite graph of the Java Class Generator	176

7.1	Approach for constructing code generators for EMF with Genesys	178
7.2	EMF workflow	179
7.3	Overview of the Ecore metamodel [Ecl05]	181
7.4	Example metamodel in Ecore: Simple taxonomy	182
7.5	Model Generate EMF SIBs	183
7.6	Model Generate Code for Model Element	184
7.7	Model Generate Code for Structural Features	185
7.8	SIBs generated from the “Simple taxonomy” metamodel	186
7.9	Taxonomy POJO Generator main model	187
7.10	Model Generate Code for Object	187
7.11	Example taxonomy model (“Media Catalog”) and generated POJO	188
7.12	Modeling on different metalevels	189
7.13	Code generators of the case study, by metalevels	190
8.1	Integration concept for combining jABC’s code generators and AndroMDA	194
8.2	Code generation approach in AndroMDA	195
8.3	jABC models for the Multiple Sequence Alignment (MSA) process	197
8.4	UML diagrams for the MSA web application	199
8.5	Associating SLGs with UML diagrams (left hand side) and configuring the AndroMDA SIB (right hand side)	200
8.6	Screens of the Generated MSA Web Application	202

Contents

Part I Motivation and Fundamentals

1	Introduction	3
1.1	Requirements of the Genesys Approach	5
1.2	Organization of the Book	9
2	The State of the Art in Code Generation	11
2.1	Influences of Compiler Construction	11
2.2	Models, Metamodels and Domain-Specific Languages	13
2.3	The Role of Code Generation	19
2.3.1	Computer-Aided Software Engineering	19
2.3.2	Generative Programming	20
2.3.3	Model Driven Architecture	22
2.3.4	Domain-Specific Modeling	24
2.3.5	Language Workbenches	25
2.3.6	Approaches without Code Generation	26
2.4	Code Generation Techniques	27
2.4.1	Programming the Code Generator	28
2.4.2	Template-Based Code Generation	29
2.4.3	Rule-Based Transformation	31
2.4.4	Round-Trip Engineering versus Full Code Generation	32
2.5	Quality Assurance of Code Generators	34
2.6	Classification of Genesys	35
3	Extreme Model-Driven Development and jABC	39
3.1	Extreme Model-Driven Development	39
3.2	jABC	43
3.2.1	Service Independent Building Blocks	46
3.2.2	Service Logic Graphs	50
3.2.3	Plugins	54

3.3	Model Execution with the Tracer	57
3.3.1	Execution Semantics	57
3.3.2	Execution Context	58
3.3.3	Control SIBs	60
3.3.4	Tracer Plugin	62
3.4	Model Checking with GEAR	63
3.4.1	Specification of Global Constraints	63
3.4.2	GEAR Plugin	66
3.5	jABC as a Basis for Realizing the Genesys Approach	68

Part II The Genesys Framework and Case Studies

4	The Genesys Framework	75
4.1	Services for Building Code Generators	78
4.1.1	Contributions to the Common SIBs	78
4.1.2	Type Mapping	79
4.1.3	Identifier Generation	83
4.1.4	Variant Management	85
4.2	Simple Example: Documentation Generator	87
4.2.1	Structuring the Generation Process	88
4.2.2	The Initialization Phase	89
4.2.3	The Generation Phase	90
4.2.4	Finalizing the Generator	94
4.2.5	General Remarks on the Example	95
4.3	Genesys Tooling	96
4.3.1	Developer Tools	96
4.3.2	User Tools	99
5	Case Studies: Code Generators for jABC	101
5.1	Bootstrapping: Java Class Extruder	102
5.1.1	The Extruder Concept	103
5.1.2	Development of the Generator	105
5.1.3	Evaluation	107
5.2	Java Class Generator	109
5.2.1	Handling Service Calls	109
5.2.2	From Single Class to Multiple Classes	112
5.2.3	Data Type Mappings and Execution Context	113
5.2.4	Variant 1: Structured Code	114
5.2.5	Variant 2: The Interpreter Approach	121
5.2.6	Genesys Code Generator Generator	125
5.2.7	Remarks on Different Versions	126

5.3	Comparison and Evaluation of the Java Code Generators .	127
5.3.1	Code Generator Models	127
5.3.2	Code Generator Results	132
5.3.3	Conclusions.....	135
5.4	Further Code Generators for jABC	136
5.4.1	Servlet Extruder and Servlet Generator	138
5.4.2	SIB Extruder and SIB Generator	139
5.4.3	Web Service Generator	140
5.4.4	leJOS and NXC Generator	140
5.4.5	BPEL Generator	142
5.4.6	C# Generator	143
5.4.7	JME Generator	144
5.4.8	EE Process Definition Generator	146
5.4.9	JML Extension for Java Class Generator	147
5.4.10	iPhone Generator	148
5.4.11	Code Generators for Ruby, Perl and JavaScript	150
5.4.12	Robocode Generator	151
6	Verification & Validation of Code Generators	155
6.1	Local Constraints for Code Generators	157
6.2	Global Constraints for Code Generators	158
6.2.1	FormulaBuilder	158
6.2.2	The Constraint Library	160
6.2.3	Occurrence Constraints	161
6.2.4	Order Constraints	163
6.2.5	Deriving Patterns & Composing Constraints	165
6.3	Testing of Code Generators	168
6.3.1	Testing the jABC Code Generators	170
6.3.2	Generation of Test Scripts for JUnit	173
7	Case Study: Domain-Specific Code Generators for EMF	177
7.1	Eclipse Modeling Framework	179
7.2	The Ecore Metamodel.....	180
7.3	EMF SIB Generator	182
7.4	Example: Taxonomy POJO Generator	185
7.5	Evaluation.....	188
8	Case Study: Service-Oriented Combination of Code Generation Frameworks	193
8.1	AndroMDA	195
8.2	Example Application: Multiple Sequence Alignment (MSA).....	196

8.3	Integrated Modeling	198
8.4	Integrated Code Generation	200
8.5	Evaluation.....	202

Part III Conclusions and Future Work

9	Conclusions.....	207
9.1	Requirements of the Genesys Approach Revisited	207
9.2	Further Applications of Genesys	210
10	Future Work.....	215
	Bibliography.....	223
	Index.....	243