Generative Programming

Methods, Tools, and Applications

Krzysztof Czarnecki and Ulrich W. Eisenecker
Contents

Chapter 1 What Is This Book About? 1
1.1 From Handcrafting to Automated Assembly Lines 1
1.2 Generative Programming 5
1.3 Benefits and Applicability 13

Part I Analysis and Design Methods and Techniques 17

Chapter 2 Domain Engineering 19
2.1 Why Is This Chapter Worth Reading? 19
2.2 What Is Domain Engineering? 20
2.3 Domain Analysis 23
2.4 Domain Design and Domain Implementation 24
2.5 Application Engineering 30
2.6 Product-Line Practices 30
2.7 Key Domain Engineering Concepts 32
  2.7.1 Domain 32
  2.7.2 Domain Scope and Scoping 34
  2.7.3 Relationships between Domains 37
  2.7.4 Features and Feature Models 38
  2.7.5 Method Tailoring and Specialization 43
2.8 Survey of Domain Analysis and Domain Engineering Methods 44
  2.8.1 Feature-Oriented Domain Analysis (FODA) 44
    2.8.1.1 FODA Process 45
  2.8.2 Organization Domain Modeling (ODM) 46
    2.8.2.1 The ODM Process 48
  2.8.3 Draco 48
  2.8.4 Capture 51
  2.8.5 Domain Analysis and Reuse Environment (DARE) 51
4.4.2 Other Information Associated with Feature Diagrams in a Feature Model 96
4.4.3 Assigning Priorities to Variable Features 99
4.4.4 Availability Sites, Binding Sites, and Binding Modes 100
  4.4.4.1 Sites 100
  4.4.4.2 Availability Sites 100
  4.4.4.3 Binding Sites and Binding Modes 101
4.4.4.4 Relationship between Optimizations and Availability Sites, Binding Sites, and Binding Modes 101
4.5 Relationship between Feature Diagrams and Other Modeling Notations and Implementation Techniques 102
  4.5.1 Single Inheritance 105
  4.5.2 Multiple Inheritance 107
  4.5.3 Parameterized Inheritance 108
  4.5.4 Static Parameterization 111
  4.5.5 Dynamic Parameterization 112
4.6 Implementing Constraints 113
4.7 Tool Support for Feature Models 114
4.8 Frequently Asked Questions about Feature Diagrams 116
4.9 Feature Modeling Process 119
  4.9.1 How to Find Features 119
  4.9.2 Role of Variability in Modeling 123
    4.9.2.1 Separation of Concerns 123
    4.9.2.2 Decomposition Techniques 125
    4.9.2.3 Variability in Modeling 128
Chapter 5 The Process of Generative Programming 131
  5.1 Why Is This Chapter Worth Reading? 131
  5.2 Generative Domain Models 131
  5.3 Main Development Steps in Generative Programming 134
  5.4 Adapting Domain Engineering for Generative Programming 135
  5.5 Domain-Specific Languages 137
  5.6 DEMRAL: Example of a Domain Engineering Method for Generative Programming 142
  5.7 Outline of DEMRAL 144
  5.8 Domain Analysis 146
    5.8.1 Domain Definition 146
    5.8.2 Domain Modeling 148
      5.8.2.1 Identification of Key Concepts 149
      5.8.2.2 Feature Modeling 151
7.4 C++ Support for Static Configuration 213
7.4.1 Static Typing 214
7.4.2 Static Binding 214
7.4.3 Inlining 214
7.4.4 Templates 218
7.4.5 Parameterized Inheritance 221
7.4.6 typedefs 221
7.4.7 Member Types 221
7.4.8 Nested Classes 222
7.5 Prohibiting Certain Template Instantiations 222
7.6 Static versus Dynamic Parameterization 224
7.7 Wrappers Based on Parameterized Inheritance 231
7.8 Template Method Based on Parameterized Inheritance 231
7.9 Parameterizing Binding Mode 234
7.10 Consistent Parameterization of Multiple Components 236
7.11 Static Interactions between Components 236
7.11.1 Components with Influence 238
7.11.2 Components under Influence 240
7.11.3 Structured Configurations 243
7.11.4 Recursive Components 245
7.11.5 Intelligent Configuration 248

Chapter 8 Aspect-Oriented Programming 251
8.1 Why Is This Chapter Worth Reading? 251
8.2 What Is Aspect-Oriented Programming? 252
8.3 Aspect-Oriented Decomposition Approaches 254
8.3.1 Subject-Oriented Programming 254
8.3.2 Composition Filters 256
8.3.3 Demeter / Adaptive Programming 259
8.3.4 Aspect-Oriented Programming 262
8.4 How Aspects Arise 264
8.5 Composition Mechanisms 267
8.5.1 Requirements on Composition Mechanisms 267
8.5.1.1 Minimal Coupling 267
8.5.1.2 Different Binding Times and Modes 270
8.5.1.3 Noninvasive Adaptability 272
8.5.2 Example: Synchronizing a Bounded Buffer 275
8.5.3 “Tangled” Synchronized Stack 278
8.5.4 Separating Synchronization Using Design Patterns 281
8.5.5 Separating Synchronization Using SOP 287
8.5.6 Some Problems with Design Patterns and Some Solutions 292
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5.6.1 Object Schizophrenia</td>
<td>294</td>
</tr>
<tr>
<td>8.5.6.2 Preplanning Problem</td>
<td>295</td>
</tr>
<tr>
<td>8.5.6.3 Traceability Problem</td>
<td>295</td>
</tr>
<tr>
<td>8.5.7 Implementing Noninvasive, Dynamic Composition in Smalltalk</td>
<td>296</td>
</tr>
<tr>
<td>8.5.7.1 Model of the Composition</td>
<td>296</td>
</tr>
<tr>
<td>8.5.7.2 Composition API</td>
<td>297</td>
</tr>
<tr>
<td>8.5.7.3 Instance-Specific Extension Protocol</td>
<td>298</td>
</tr>
<tr>
<td>8.5.7.3.1 Defining Methods in Instances</td>
<td>299</td>
</tr>
<tr>
<td>8.5.7.3.2 Adding Instance Variables to Instances</td>
<td>301</td>
</tr>
<tr>
<td>8.5.8 Kinds of Crosscutting</td>
<td>303</td>
</tr>
<tr>
<td>8.6 How to Express Aspects in Programming Languages</td>
<td>305</td>
</tr>
<tr>
<td>8.6.1 Separating Synchronization Using AspectJ Cool</td>
<td>306</td>
</tr>
<tr>
<td>8.6.2 Implementing Dynamic Cool in Smalltalk</td>
<td>310</td>
</tr>
<tr>
<td>8.6.2.1 Example: Synchronizing an Assembly System</td>
<td>310</td>
</tr>
<tr>
<td>8.6.2.2 Architecture of the Smalltalk Implementation of Dynamic Cool</td>
<td>315</td>
</tr>
<tr>
<td>8.7 Implementation Technologies for Aspect-Oriented Programming</td>
<td>317</td>
</tr>
<tr>
<td>8.7.1 Technologies for Implementing Aspect-Specific Abstractions</td>
<td>317</td>
</tr>
<tr>
<td>8.7.2 Technologies for Implementing Weaving</td>
<td>321</td>
</tr>
<tr>
<td>8.7.3 AOP and Specialized Language Extensions</td>
<td>328</td>
</tr>
<tr>
<td>8.7.4 AOP and Active Libraries</td>
<td>328</td>
</tr>
<tr>
<td>8.8 Final Remarks</td>
<td>331</td>
</tr>
<tr>
<td>Chapter 9 Generators</td>
<td>332</td>
</tr>
<tr>
<td>9.1 Why Is This Chapter Worth Reading?</td>
<td>332</td>
</tr>
<tr>
<td>9.2 What Are Generators?</td>
<td>333</td>
</tr>
<tr>
<td>9.3 Transformational Model of Software Development</td>
<td>335</td>
</tr>
<tr>
<td>9.4 Technologies for Building Generators</td>
<td>339</td>
</tr>
<tr>
<td>9.5 Compositional versus Transformational Generators</td>
<td>341</td>
</tr>
<tr>
<td>9.6 Kinds of Transformations</td>
<td>344</td>
</tr>
<tr>
<td>9.6.1 Compiler Transformations</td>
<td>344</td>
</tr>
<tr>
<td>9.6.1.1 Refinements</td>
<td>344</td>
</tr>
<tr>
<td>9.6.1.2 Optimizations</td>
<td>345</td>
</tr>
<tr>
<td>9.6.2 Source-to-Source Transformations</td>
<td>348</td>
</tr>
<tr>
<td>9.7 Transformation Systems</td>
<td>350</td>
</tr>
<tr>
<td>9.7.1 Scheduling Transformations</td>
<td>352</td>
</tr>
<tr>
<td>9.7.2 Existing Transformation Systems and Their Applications</td>
<td>355</td>
</tr>
</tbody>
</table>
9.8 Selected Approaches to Generation
9.8.1 Draco 357
9.8.2 GenVoca 363
   9.8.2.1 Example: Applying GenVoca to the Domain of Data Containers 364
   9.8.2.2 GenVoca Model 366
   9.8.2.3 Defining Realms and Layers in P++ 372
   9.8.2.4 Implementing GenVoca Layers in C++ 375
   9.8.2.5 Composition Validation 385
   9.8.2.6 Transformations and GenVoca 387
   9.8.2.7 Frameworks and GenVoca 388
9.8.3 Approaches Based on Algebraic Specifications 389

Chapter 10 Static Metaprogramming in C++ 397
10.1 Why Is This Chapter Worth Reading? 397
10.2 What Is Metaprogramming? 398
10.3 A Quick Tour of Metaprogramming 399
10.4 Static Metaprogramming 405
10.5 C++ As a Two-Level Language 406
10.6 Functional Flavor of the Static Level 410
   10.6.1 Class Templates As Functions 410
   10.6.2 Integers and Types As Data 411
   10.6.3 Symbolic Names Instead of Variables 411
   10.6.4 Constant Initialization and typedef-Statements Instead of Assignment 411
   10.6.5 Template Recursion Instead of Loops 412
   10.6.6 Conditional Operator and Template Specialization As Conditional Constructs 413
10.7 Template Metaprogramming 413
10.8 Template Metafunctions 414
10.9 Metafunctions As Arguments and Return Values of Other Metafunctions 416
10.10 Representing Metainformation 418
   10.10.1 Member Traits 419
   10.10.2 Traits Classes 420
   10.10.3 Traits Templates 421
   10.10.4 Example: Using Template Metafunctions and Traits Templates to Implement Type Promotions 425
   10.10.5 Compile-Time Lists and Trees As Nested Templates 427
10.11 Compile-Time Control Structures 433
10.11.1 Explicit Selection Constructs (? :, IF<>), and SWITCH<>)
10.11.1.1 Implementing IF<> without Partial Template Specialization 433
10.11.1.2 Switch Construct (SWITCH<>) 436
10.11.1.3 Taking Advantage of Lazy Behavior 438
10.11.1.4 SWITCH<> without Partial Template Specialization 444
10.11.2 Template Recursion As a Looping Construct 444
10.11.3 Explicit Looping Constructs (WHILE<>, DO<>), and FOR<>)
10.11.3.1 While-Loop (WHILE<>) 451
10.11.3.2 Do-Loop (DO<>) 455
10.11.3.3 For-Loop (FOR<>) 459
10.12 Code Generation 462
10.12.1 Simple Code Selection 462
10.12.2 Composing Templates 464
10.12.3 Generators Based on Expression Templates 464
10.12.4 Recursive Code Expansion 465
10.12.5 Explicit Loops for Generating Code (EWHILE<>, EDO<>, and EFOR<>) 473
10.12.5.1 EWHILE<> 473
10.12.5.2 EDO<> 476
10.12.5.3 EFOR<> 477
10.12.5.4 Nesting Static E-Loops 479
10.13 Example: Using Static Execute Loops to Test Metafunctions 487
10.14 Partial Evaluation in C++ 493
10.15 Workarounds for Partial Template Specialization 497
10.16 Problems of Template Metaprogramming 499
10.17 Historical Notes 499

Chapter 11 Intentional Programming 503
11.1 Why Is This Chapter Worth Reading? 503
11.2 What Is Intentional Programming? 504
11.3 Technology behind IP 510
11.3.1 System Architecture 510
11.3.2 Representing Programs in IP: The Source Graph 512
11.3.2.1 Treelike Structures 513
11.3.2.2 Graphlike Structures 513
11.3.2.3 Source Graphs: The Big Picture 516
11.3.2.4 The Essence of Source Graphs: Abstraction Sharing and Parameterization 518
11.3.3 Source Graph + Methods = Active Source 518
  11.3.3.1 Kinds of Methods 519
11.4 Working with the IP Programming Environment 522
  11.4.1 Editing 524
  11.4.2 Further Capabilities of the IP Editor 528
  11.4.3 Extending the IP System with New Intentions 537
11.5 Advanced Topics 541
  11.5.1 Questions, Methods, and a Frameworklike Organization 542
  11.5.2 Source-Pattern-Based Polymorphism 543
  11.5.3 Methods As Visitors 545
  11.5.4 Asking Questions Synchronously and Asynchronously 546
  11.5.5 Reduction 547
11.6 The Philosophy behind IP 551
  11.6.1 Why Do We Need Extendible Programming Environments? or What Is the Problem with Fixed Programming Languages? 551
    11.6.1.1 Problems with General-Purpose Languages and Conventional Libraries 551
    11.6.1.2 Problems with Comprehensive Application-Specific Languages 553
    11.6.1.3 The IP Solution: An Extendible Programming Environment 555
  11.6.2 Moving Focus from Fixed Languages to Language Features and the Emergence of an Intention Market 556
  11.6.3 Intentional Programming and Component-Based Development 557
  11.6.4 Frequently Asked Questions 559
11.7 Summary 566

Part III Application Examples 569
Chapter 12 List Container 571
  12.1 Why Is This Chapter Worth Reading? 571
  12.2 Overview 571
  12.3 Domain Analysis 572
  12.4 Domain Design 573
  12.5 Implementation Components 577
  12.6 Manual Assembly 583
  12.7 Specifying Lists 586
  12.8 The Generator 586
  12.9 Extensions 590
Chapter 13 Bank Account

13.1 Why Is This Chapter Worth Reading? 593
13.2 The Successful Programming Shop 593
13.3 Design Patterns, Frameworks, and Components 596
13.4 Domain Engineering and Generative Programming 597
13.5 Feature Modeling 599
13.6 Architecture Design 600
13.7 Implementation Components 603
13.8 Configurable Class Hierarchies 610
13.9 Designing a Domain-Specific Language 612
13.10 Bank Account Generator 619
13.11 Testing Generators and Their Products 623

Chapter 14 Generative Matrix Computation
Library (GMCL) 624

14.1 Why Is This Chapter Worth Reading? 624
14.2 Why Matrix Computations? 625
14.3 Domain Analysis 626
14.3.1 Domain Definition 626
14.3.2 Domain Modeling 628
14.4 Domain Design and Implementation 634
14.4.1 Matrix Type Generation 638
14.4.1.1 Target Architecture and Matrix Implementation Components 639
14.4.1.2 Matrix Configuration DSL 654
14.4.1.3 Matrix Configuration Generator 661
14.4.1.3.1 Configuration DSL Parser 664
14.4.1.3.2 Assigning Defaults to DSL Features 666
14.4.1.3.3 Matrix Component Assembler 672
14.4.2 Generating Code for Matrix Expressions 678
14.4.2.1 Evaluating Matrix Expressions 678
14.4.2.2 Modeling the Elementwise Expression Evaluation Using Objects 680
14.4.2.3 Overview of the Implementation 685
14.4.2.4 Matrix Operator Templates 686
14.4.2.5 Matrix Expression Templates 690
14.4.2.6 Matrix Cache 696
14.4.2.7 Generating getElement 698
14.4.2.8 Metafunctions for Computing Result Types of Expressions 701
14.4.2.9 Generating Matrix Assignment 706
### Contents

14.4.2.10 Lessons Learned 708  
14.4.2.11 Problems with Template Metaprogramming 710  
14.4.3 Implementing the Matrix Component in IP 713

**Appendices** 719

Appendix A Conceptual Modeling 721

A.1 What Are Concepts? 721  
A.2 Theories of Concepts 722  
A.2.1 Basic Terminology 723  
A.2.2 The Classical View 724  
A.2.3 The Probabilistic View 727  
A.2.4 The Exemplar View 728  
A.2.5 Summary of the Three Views 728  
A.3 Important Issues Concerning Concepts 730  
A.3.1 Stability of Concepts 730  
A.3.2 Concept Core 731  
A.3.3 Informational Contents of Features 732  
A.3.4 Feature Composition and Relationships between Features 732  
A.3.5 Quality of Features 733  
A.3.6 Abstraction and Generalization 733  
A.4 Conceptual Modeling, Object-Orientation, and Software Reuse 736

Appendix B Instance-Specific Extension Protocol for Smalltalk 738

Appendix C Protocol for Attaching Listener Objects in Smalltalk 741

Appendix D Glossary of Matrix Computation Terms 746

Appendix E Metafunction for Evaluating Dependency Tables 749

Glossary of Generative Programming Terms 754

References 757

Index 799