

Contents

Preface	iii
1 Introduction	1
1.1 Generators	3
1.2 Domain-Specific Languages	6
1.3 Reuse	9
I How Generators Provide Leverage	15
2 Structured Output	19
2.1 The Web site Maintenance Problem	20
2.2 Generating the Memex Web Site Text Module	24
2.2.1 An abstraction of structured text	24
2.2.2 Building a structured text abstraction	29
2.2.3 Converting to linear text	31
2.3 Programming the Memex Web Site Text Module	34
2.3.1 Class definitions for Web page text fragments	34
2.3.2 Constructors for Web page text fragments	36
2.4 Hand Coding vs. Generation	37
3 Storing Entities with Properties	39
3.1 The Student Data Problem	40
3.2 Generating a Property Storage Module	42
3.3 Programming a Property Storage Module	45
3.3.1 A general property module	45
3.3.2 Implementing property types	48
3.3.3 Implementing access mechanisms	48
3.3.4 Applying access mechanisms to properties	50
3.4 Hand Coding vs. Generation	51

4	Visiting Trees	55
4.1	The Table Layout Problem	56
4.2	Generating the Table Layout Module	60
4.2.1	Specifying the tree structure	60
4.2.2	Specifying computations in trees	61
4.2.3	The generated visitor module	65
4.3	Programming the Table Layout Module	65
4.3.1	Implementing the tree structure	66
4.3.2	Using the Visitor design pattern	68
4.3.3	Scheduling operations on tree walks	72
4.3.4	Storing values during tree walks	75
4.4	Hand Coding vs. Generation	77
5	Names and Entities	81
5.1	The Structure Generator Task	82
5.2	Specifying Name Analysis for the Structure Generator	83
5.2.1	Associating name analysis roles with language constructs	83
5.2.2	Name analysis computations	87
5.3	Programming the Name Analysis Phase	90
6	Constructing Trees	95
6.1	The Calendar Tree Construction Problem	95
6.2	Generating a Calendar Structuring Program	98
6.2.1	Generating a parser	99
6.2.2	Generating a lexical analyzer	100
6.2.3	Generating a tree builder	103
6.3	Programming a Calendar Structuring Program	104
6.3.1	Programming a lexical analyzer	105
6.3.2	Programming a parser	109
6.3.3	Programming a tree builder	112
6.4	Hand Coding vs. Generation	112
7	An Integrated Approach	115
7.1	Task Decomposition	115
7.2	Generating a Structure Generator	117
7.2.1	Structural analysis	117
7.2.2	Specification of computations in tree contexts	120
7.2.3	Property analysis	122
7.2.4	Transformation	125

7.3	Programming a Structure Generator	129
7.3.1	Structural analysis	130
7.3.2	Semantic analysis	131
7.3.3	Transformation	133
7.4	Hand Coding vs. Generation	134
II	Support Infrastructure	139
8	Execution Monitoring	141
8.1	Source-Level Execution Models	142
8.2	Specification-Level Execution Models	144
8.3	The Noosa Execution Monitor	146
8.4	Debugging Situations	151
8.5	Implementing Execution Monitors	153
9	Modularity and Generators	157
9.1	A File-Based Module Mechanism	158
9.1.1	Genericity	159
9.1.2	Influence on specification language design	160
9.2	Literate Programming	161
9.2.1	Why use literate programming?	165
9.2.2	Design strategies for specification-based programs	166
9.3	FunnelWeb Specifications	167
10	Software Manufacturing	171
10.1	Two Manufacturing Scenarios	173
10.1.1	Produce a structure generator	173
10.1.2	Monitor execution	175
10.2	Defining a Manufacturing Process	177
10.2.1	Managing the overall process	177
10.2.2	Carrying out a single manufacturing process	183
10.2.3	Defining common infrastructure	188
10.3	Implementing the Process	193
10.3.1	Process-independent operations	193
10.3.2	Process-dependent operations	196

III Complete Examples	203
11 Desk Calculator	205
11.1 Expressions	206
11.2 Expression Values	210
11.3 Extensions	213
11.3.1 More than one expression	213
11.3.2 Variables	215
11.3.3 Constants	219
11.3.4 Library functions	220
11.4 Lessons Learned	226
12 Language Extension	229
12.1 Extending Java with Formal Specifications	229
12.1.1 A Java extension	230
12.1.2 Implementation strategy	233
12.2 Implementing Language Extensions	234
12.3 Unparsing	236
12.3.1 Deriving an unparser	237
12.3.2 Idem unparsing	237
12.3.3 Unparsing terminal symbols	239
12.3.4 Overriding node representations	239
12.4 Implementing the Java Extension	240
12.4.1 Defining the tree structure	240
12.4.2 Translation	242
12.4.3 Completing the specification	249
12.4.4 Specification files	255
12.5 Lessons Learned	256
13 A Pattern-based Text Generator	259
13.1 Structuring	260
13.2 Name Analysis	262
13.2.1 Insertion points	263
13.2.2 Pattern names	265
13.2.3 Function calls	272
13.3 Property Analysis	273
13.3.1 Types of insertion points	273
13.3.2 Type check for function calls	275

13.4	Transformation	280
13.4.1	General techniques	280
13.4.2	Insertion key lists	281
13.4.3	Special case conditions	287
13.4.4	Generating the factory module	291
13.4.5	Factory function implementation	294
13.4.6	Generating the pattern class module	300
13.4.7	Implementation of the <code>Write</code> methods	307
14	A Manufacturing Problem	315
14.1	The Document Manufacturing Process	316
14.1.1	Raw material	316
14.1.2	Tools	318
14.1.3	Relationships	319
14.2	Automating the Process	320
14.2.1	Describing the document manufacturing problem	321
14.2.2	Determining the formatter and BibTeX inputs	327
14.2.3	Format the text	338
14.2.4	Format the bibliography	348
14.2.5	Update the auxiliary files	350
14.2.6	Obtain PostScript text	351
14.2.7	Obtain PDF text	353
14.3	The TeX Package	353
14.4	A Filter for Included Filenames	357
14.4.1	Interesting TeX commands	357
14.4.2	Check for an interesting command	360
14.4.3	<code>texscan.c</code>	362
	Bibliography	365
	Index	369