I Foundations 1

1 Introduction 3

1.1 Entity-Life Modeling 3
   1.1.1 Reactive Systems 5

1.2 Overview of This Book 5

1.3 Multithreading 6
   *1.3.1 Preemptive and Nonpreemptive Threading 7
   1.3.2 Using Threads 7
      1.3.2.1 Thread Scheduling 8
      1.3.2.2 Message Passing 8
      1.3.2.3 A Different Mindset 9

1.4 Engineering the Intangible 9
   1.4.1 Software Architecture 10
      1.4.1.1 Thread Architecture 10
   1.4.2 Conceptual Integrity 11
      1.4.2.1 The Key Idea of an Architecture 12
*1.4.3 Analogical Modeling 12

1.5 The Development Process 13
   1.5.1 ELM in the Development Process 14
   1.5.2 Analysis and Design 15
   1.5.3 Design Upfront 15
      1.5.3.1 Refactoring 17
2.3.1.1 Declaration of Single Tasks and Task Types 52
2.3.1.2 Instantiation of Task Types 52
2.3.1.3 Task Activation 53
2.3.1.4 Task Priorities 54
2.3.1.5 The delay Statement 54
2.3.1.6 Exceptions in Tasks 55

2.3.2 Protected Objects 55
  *2.3.2.1 Protected Interfaces 57
  2.3.2.2 Requeuing 58
  2.3.2.3 Conditional and Timed Entry Calls 59
  2.3.2.4 Interrupt Handling 59
  2.3.2.5 Timing Events 60
  2.3.2.6 Controlling Access to Shared Domain Resources 61

2.3.3 Asynchronous Transfer of Control 62
  2.3.3.1 Example: Bumping an FMS Job 63

*2.3.4 Rendezvous 64

2.4 Pthreads 65
  2.4.1 Managing Pthreads 65
    2.4.1.1 Example of Pthreads 66
  2.4.2 Mutex Variables and Exclusion Synchronization 66
    2.4.2.1 Example of Pthreads and Mutexes 66
  2.4.3 Condition Variables and Condition Synchronization 67

2.4.4 Pthreads: Conclusion 68

2.5 Conclusion 68

Exercises 68

3 State Modeling 71

3.1 Introduction 71
  3.1.1 State Modeling and Object Orientation 72

3.2 State-Modeling Terminology 72
  3.2.1 States and Events 73
    3.2.1.1 Time Events 74
    *3.2.1.2 Determinism 74

3.3 Basic State Modeling 75
  3.3.1 A Simple Example 75
  3.3.2 Guard Conditions 76
    *3.3.2.1 Guard Conditions Based on Modeled-Entity Attributes 77
  3.3.2.2 Ambient Guard Conditions 78
  3.3.2.3 Complex Guard Conditions 78
  3.3.2.4 Example: Traffic Light 78
    *3.3.2.5 State Machines Associated with Aggregate Entities 79

3.3.3 State Tables 79

3.3.4 Actions and Activities 80
  3.3.4.1 Actions 80
  3.3.4.2 Activities 82
  3.3.4.3 Actions or Activities? 83
4.2.1 ELM Rationale 111
  4.2.1.1 Object-Oriented Modeling 111
  4.2.1.2 Modeling in the Time Dimension 112
  4.2.1.3 Time Events 113
  *4.2.1.4 Events Shared by Problem and Software Domains 113
4.2.2 Event Threads and Event-Thread Models Defined 114
  4.2.2.1 Data-Entry Example 115
  *4.2.2.2 Impractical and Counterintuitive Event-Thread Models 116
  *4.2.2.3 Exceptional Events in a Thread 116
4.2.3 Concurrency Levels and Optimal Event-Thread Models 117
  4.2.3.1 Coincidental Simultaneity 118
  4.2.3.2 Concurrency Levels and Optimality 119
  4.2.3.3 Nonoptimal Event-Thread Models 119
  *4.2.3.4 Multiprocessors 120
4.2.4 Latitude for the Designer 120
  4.2.4.1 Accidental Constraints 121
  4.2.4.2 Many Simultaneous Occurrences 121
4.2.5 Design Based on Event Threads 121
  4.2.5.1 Design Patterns for Implementing Threads 122
4.3 Discovering and Choosing Event-Thread Models 123
  4.3.1 Identifying Individual Entities and Event Threads 123
    4.3.1.1 Operator Threads 123
    4.3.1.2 Periodic Threads 124
    4.3.1.3 Long-Lived Threads 125
  4.3.2 Example of Thread Identification: Elevator System 125
4.4 Event-Thread Patterns for Resource Sharing 128
  4.4.1 Simultaneous Exclusive Access to Multiple Resources 129
  4.4.2 Example: Jukebox 130
    4.4.2.1 Resource-User-Thread Model: A Thread per Customer 130
    4.4.2.2 Resource-Guard-Thread Model: A Thread per Panel 130
    4.4.2.3 Comparison of Thread Models and Architectures 130
  4.4.3 Example: Queuing System for a Bank Office 131
    4.4.3.1 Resource-User-Thread Model: A Thread per Customer 131
    4.4.3.2 Resource-Guard-Thread Model: A Thread per Teller 132
    4.4.3.3 Comparison of Thread Models and Designs 132
  *4.5 Portraying the World in Software 133
4.6 Conclusion 134
Appendix 4A: Summary of Terms 135
Exercises 136

5 Design Patterns Based on Event Threads 139
  5.1 Introduction 139
    5.1.1 Software Activities and Nominal Activities 140
6 Event-Thread Patterns for Resource Sharing  173

6.1 Introduction  173
  6.1.1 Duality of the Patterns  174

6.2 Resource-User-Thread Pattern  174
  6.2.1 Exclusive Access to Domain Objects  175
    6.2.1.1 Implementation of Semaphores for Domain Objects  175
    6.2.1.2 Implementation of Monitors for Domain Objects  177
  6.2.2 Programming Style  177

6.3 The Resource-Guard-Thread Pattern  177
  6.3.1 Queuing  178
  6.3.2 Resource-Independent Processing  178
    *6.3.3 Guard Threads Implemented as Concurrent Activities  178

6.4 Choosing and Combining Patterns  179
6.4.1 Resource-Guard Threads Doubling as Resource Users 179
*6.4.1.1 One Resource-User Event Thread—A Series of Control Threads 180
6.4.2 Choosing Resource-User or Resource-Guard Threads 180
6.5 Examples with Dual Solutions 181
6.5.1 Remote Temperature Sensor 181
  6.5.1.1 RTS: Resource-User-Thread Solution 182
  6.5.1.2 RTS: Resource-Guard-Thread Solution 183
  6.5.1.3 RTS: Comparison of the Solutions 183
6.5.2 Home-Heating System 183
  6.5.2.1 Home Heater: Resource-User-Thread Solution 184
*6.5.2.2 Home Heater: Dual Solution 185
6.5.3 Automated Store 186
  6.5.3.1 Resource-User-Thread Solution 186
  6.5.3.2 Resource-Guard-Thread Solution 187
6.6 Data Stream Processing 188
  6.6.1 Surveillance Radar Problem 188
  6.6.2 MIDI Problem 189
    6.6.2.1 Programmable Patch Bay 189
6.7 Repository Problems 190
  6.7.1 Multielevator System 190
    6.7.1.1 Solution Sketch 191
    6.7.1.2 Concurrency Levels in the Elevator Problem 192
  6.7.2 Traffic Light System 192
  6.7.3 Repository Problem Solutions 192
6.8 Conclusion 193
Exercises 194

7 Simultaneous Exclusive Access to Multiple Resources 197
7.1 Introduction 197
7.2 The Deadlock Problem 198
  7.2.1 Determining That a System is Deadlock Free 199
  7.2.2 Deadlock Prevention 201
    7.2.2.1 Resource Ordering 202
    7.2.2.2 Limiting the Number of Entities 202
    7.2.2.3 Avoiding Indefinite Waiting 203
  7.2.3 Dining Philosophers’ Problem 205
    7.2.3.1 Deadlock Prevention in the Philosophers’ Problem 205
7.3 Case Studies 206
  7.3.1 Automated Train Switchyard 206
    7.3.1.1 Deadlock Analysis of the Switchyard 208
    7.3.1.2 Optimization 209
*7.3.1.3 Realism 209
    7.3.1.4 Hump Yards 210
  7.3.2 Flexible Manufacturing System 210
    7.3.2.1 Deadlock Prevention in the FMS 212
7.3.2.2 Job-Thread Solution for the FMS 213
7.3.2.3 Workstation-Thread Solution for the FMS 215
7.3.2.4 Other FMS Solutions 217
7.3.3 AGV System Simulation 218
   7.3.3.1 Solution Sketch for the AGV System 218
   7.3.3.2 Complications 219
7.4 Heuristics 220
   7.4.1 Entities Driving the Process 220
      7.4.1.1 Anthropomorphism? 221
7.5 More on Deadlock and Its Prevention 221
   7.5.1 Deadlock with and without Threads 221
   7.5.2 Deadlock Involving Internal Software Resources 222
   7.5.3 Expanding an ELM Architecture 222
   7.5.4 Problems without Apparent Resources 223
   7.5.5 Acquiring All Resources Ahead of Time 224
7.6 Conclusion 224
Exercises 225

III Background and Discussion

8 Real-Time Software Architectures and Data-Flow Design Approaches 231
8.1 Introduction 231
8.2 Real-Time Architectures 232
   8.2.1 Cyclic Executive 232
      8.2.1.1 Cyclic Executive Implementations with Threads 233
   8.2.2 Periodic Threads 234
      8.2.2.1 Rate-Monotonic Scheduling 234
   8.2.3 Dynamically Scheduled Threads 236
   8.2.4 Requirements Representations versus Architectures 237
8.3 Data-Flow Design Approaches 238
   8.3.1 Structured Analysis 238
      8.3.1.1 Strengths and Weaknesses of Structured Analysis 239
      8.3.1.2 Design and Implementation Based on Data Flow 240
      8.3.1.3 Real-Time Structured Analysis 241
   8.3.2 Data-Flow Threading 242
      8.3.2.1 Mascot 242
      8.3.2.2 Data Flow and Object Orientation 243
      8.3.2.3 Advantages of Data-Flow Threading 243
      8.3.2.4 Drawbacks of Data-Flow Threading 244
   8.3.3 Example Approach: COMET 248
      8.3.3.1 Steps 1 and 2: Designing High-Level Architecture; Structuring Subsystems 249
      8.3.3.2 Step 4: Structuring the Threads 249
      8.3.3.3 Step 6: Designing Classes 250
      8.3.3.4 Step 7: Detailed Design 250
8.3.4 COMET Solution for the Cruise Controller 250
   8.3.4.1 Cruise Controller Software Architecture 251
   8.3.4.2 Thread Structuring 252
   8.3.4.3 Thread Interfaces 252
   8.3.4.4 Comparison with ELM 253

8.4 Conclusion 255

9 The Origins of Entity-Life Modeling 257
  9.1 Introduction 257
  9.2 Early Experiences with Software Development 258
     9.2.1 Systems Programming 259
     9.2.2 Multithreading 259
  9.3 The Jackson Methods 260
     9.3.1 Jackson Structured Programming 261
        9.3.1.1 Structure of Data 261
        9.3.1.2 Program Control Structure 262
        9.3.1.3 Programs Based on Combined Data Tree Diagrams 263
        9.3.1.4 Structure Clashes 264
        9.3.1.5 Real-Life JSP Example 265
        9.3.1.6 The Difficult and the Simplistic 265
     9.3.2 Implicit State Representation 266
     9.3.3 Explicit State Representation 267
        *9.3.3.1 Inversion with Respect to Event Threads 267
     9.3.4 JSD, Threading, and ELM 268
     9.3.5 Reconciling the Object and Process Models 269
  *9.4 Formal Models and Methods 270
     9.4.1 Process Algebra 270
     9.4.2 Other Formalism 270
     9.4.3 The Need for Formalism 271
     9.4.4 Concurrency in Other Languages 271
  9.5 Software Patterns 272
     9.5.1 ELM Patterns 273
        9.5.1.1 Event-Thread Patterns for Resource Sharing 273
        9.5.1.2 State Machine–Related Design Patterns 274
        *9.5.1.3 Distinction between Event-Thread and Design Patterns 274
  9.6 Conclusion 274

Exercises 275

Glossary 279
References 283
Index 293