REAL-TIME OBJECT UNIFORM DESIGN METHODOLOGY WITH UML

Real-Time Object Uniform Design Methodology with UML

BUI MINH DUC
Laval University, Sainte-Foy, QC, Canada

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN 978–1–4020–5976–6
ISBN 978–1–4020–5977–3

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

All Rights Reserved
© 2007
No part of this work may be reproduced, stored in a retrieval system, or transmitted
in any form or by any means, electronic, mechanical, photocopying, microfilming, recording
or otherwise, without written permission from the Publisher, with the exception
of any material supplied specifically for the purpose of being entered
and executed on a computer system, for exclusive use by the purchaser of the work.

Contents

List of Figures xi

List of Tables xxv

Preface xxix

1
Introduction to the world of systems, problems, and solutions 1

1.1 The world of systems, problems, and solutions 1

1.1.1 System: problem 1

1.1.2 Tools for handling complex systems 4

1.1.3 Intelligent systems: hybrid systems, multidisciplinary systems 5

1.1.4 Component-based systems 7

1.1.5 Solutions 8

1.2 Real-time and embedded systems: disaster control and quality of service 10

1.2.1 Real-time and embedded systems 10

1.2.2 Disaster control: quality of service 12

1.3 Human organizations and structures: how software can simulate and study them 13

1.3.1 Human organizations and structures 13

1.3.2 Simulation of human organizations 14

1.4 This book is also a system 15

2
Coping with complexity 19

2.1 Visual formalisms for handling complexity: a picture is worth a thousand words 19

2.1.1 Free diagrams 19

2.1.2 Technical diagrams 20

2.2 Object/function decomposition: layering, hierarchy, collaboration 22

2.3 Functional decomposition to handle complexity 23

2.3.1 Functional modeling is based on processes 23

2.3.2 How process is converted to UML? 24

2.3.3 The limit of functional decomposition: half a century of modular programming 25

2.4 How object technology handles complexity 28

2.4.1 Object paradigm 28
2.4.2 Information hiding, encapsulation 30
2.4.3 Composite objects are built from aggregates 30
2.4.4 Class factorization as complexity reduction mechanism 31
2.4.5 Unique messaging mechanism for object interaction 32
2.4.6 Inheritance as an economy of new class description 33
2.5 Object paradigm applied to control applications: reactive systems, control arrangement 33
2.6 Object paradigm applied to database applications: data modeling, handling database complexity 36
2.7 Objects and databases: bridging or mingling? 39
2.8 Object paradigm and component reuse 40
2.9 Mastering development complexity: a roadmap towards Model Driven Architecture 42

3
UML Infrastructure: Core concepts 47
3.1 Core concepts compliant to MOF: Modeling levels 48
3.2 Infrastructure: Core package 50
3.2.1 Core::PrimitiveTypes Package 52
3.2.2 Core::Abstractions Package 53
3.2.3 Core::Basic Package 65
3.2.4 Core::Constructs Package 67
3.2.5 Core::Profiles Package 72

4
UML Superstructure: language definition and diagrams 77
4.1 Making our own development methodology with UML 77
4.2 Structure and behavior: 13 UML diagrams classified along Structure and Behavior axes 79
4.3 Hierarchy of UML metaclasses 83
4.3.1 UML and "L" as language 83
4.3.2 Package import and merge in metamodeling 85
4.3.3 Comment on inheritance relationship at the metalevel of UML 86
4.4 Superstructure and compliance levels 86
4.5 Reuse of the Infrastructure: generalization hierarchy of metaclasses in the Structure part 88
4.6 Structure: class diagram 93
4.6.1 Example of user class diagram 95
4.6.2 Attribute 96
4.6.3 Association end, navigability, role, and qualifier 98
4.6.4 Operation, parameter list, return type, signature 100
4.6.5 Interface 103
4.6.6 Association class 104
4.6.7 Graphical notations 105
4.6.8 Dependencies 108
4.7 Structure: object diagram 109

4.7.1 Object, instance of a class 110
4.7.2 Link, instance of an association 111
4.7.3 Graphical notations 113
4.8 Structure: package diagram 114
4.9 Structure: composite structure diagram 118
4.9.1 Part 120
4.9.2 Port 122
4.9.3 Connector 123
4.9.4 Collaboration and collaboration use 123
4.9.5 Graphical notations 126
4.10 Structure: component diagram 128
4.11 Structure: deployment diagram 132
4.12 Hierarchy of metaclasses defined in the Behavior part: abstract metaclass behavior 135
4.13 Behavior: state machine diagram 139
4.13.1 State 142
4.13.2 State transition 143
4.13.3 Trigger and event 145
4.13.4 Simple, composite, and submachine states 145
4.13.5 Graphical notations 147
4.14 Behavior: activity diagram 150
4.14.1 Activity and Action metaclasses 150
4.14.2 Activity diagram 155
4.14.3 Data and control flow: execution of an activity diagram 158
4.14.4 Object nodes and storage of tokens 160
4.14.5 Control nodes 161
4.14.6 Exception handling: partition 163
4.14.7 Graphical notations 164
4.15 Behavior: interaction suite 169
4.15.1 Example of sequence diagram 169
4.15.2 Example of communication diagram 171
4.15.3 Example of Interaction Overview Diagram 173
4.15.4 Example of timing diagram 174
4.15.5 Metaclasses of the Interaction Suite 175
4.15.6 Graphical notations 176
4.16 Behavior: Use case diagram 181
4.16.1 Example 183
4.16.2 Metaclasses 183
4.16.3 Graphical notations 184
4.17 Auxiliary constructs: profiles 186
4.17.1 Example of profile 187
4.17.2 Graphical notations 188

5
Fundamental concepts of the real world and their mapping in UML 191
5.1 Abstraction, concept, domain, ontology, model 191
5.1.1 Abstraction 191

5.1.2 Concept 193
5.1.3 Domain 194
5.1.4 Ontology 196
5.1.5 Model 198
5.2 Structural, functional, and dynamic views of systems 200
5.3 Concepts of the functional view: Process and Business Process Modeling 201
5.3.1 Concepts used in Process Domain 202
5.3.2 Examples of Process Modeling and their conversions into UML diagrams 207
5.3.3 Building a proprietary methodology 211
5.3.4 Business process domain 211
5.3.5 Business Process Management and Workflow 216
5.3.6 Business Motivation Model 217
5.3.7 Mapping a process 220
5.3.8 Mapping a datastore 225
5.3.9 Mapping control flows and data flows 227
5.4 Fundamental concepts of the dynamic view 230
5.4.1 States and pseudostates 231
5.4.2 From elementary state towards global system state 234
5.4.3 Actions and activities 238
5.4.4 Events 240
5.4.5 Condition 241
5.4.6 Messages 245
5.4.7 Relations between action, activity, event, state, condition and message 248
5.4.8 Transitions: relationship with control flow 253
5.5 Fundamental concepts of the structural view 255
5.5.1 Class, type, object, and set of objects 258
5.5.2 Database objects, real-time objects, and their relationships 268
5.5.3 Dependencies, relationships, associations, links 270
5.5.4 Association class: n-ary association 272
5.5.5 Aggregation/composition in the object/component hierarchy 276
5.5.6 Inheritance 280
5.5.7 Multiple inheritances 281
5.5.8 Objects and roles 285

6
The Uniform concept 289
6.1 Elements of the uniform concept 290
6.1.1 Elements of the real world: objects and their interactions 290
6.1.2 Extension of the message interaction model 293
6.1.3 Induced energy 296
6.1.4 Use of a Monitor object 297
6.1.5 All objects are made independent even with tightly coupled interaction scheme 301
6.1.6 Cause–effect chain 303

6.2 Requirement analysis model 305
6.2.1 "What to do?" phase 305
6.2.2 Parameters of the requirements engineering step 306
6.2.3 Requirements engineering 308
6.3 Requirements analysis with technical diagrams 311
6.3.1 Requirements analysis with use case diagrams 313
6.3.2 Conducting the requirements analysis phase 318
6.3.3 Use case modeling advices 324
6.3.4 Using business process diagrams 326
6.4 System Requirements specifications 327
6.4.1 Requirements Engineering Model in the Model Driven Architecture 329
6.4.2 Dealing with platform dependent components in PIM 331
6.4.3 Various steps in the design process 331
6.4.4 Hierarchical decomposition or bottom-up method? 334
6.5 Designing real-time applications with MDA 335
6.5.1 Project breakdown structure into packages 337
6.5.2 Project scheduling 340
6.5.3 Starting a real-time system with the UML sequence diagram 340
6.5.4 What does a logical model mean? 344
6.5.5 Starting real-time systems with the UML interaction overview, activity, or state machine diagrams 347
6.5.6 Building real-time classes and object diagrams 350
6.5.7 Completing dynamic studies with UML state machine and activity diagrams 357
6.5.8 Assigning task responsibility 362
6.6 Designing reusable database models 364
6.6.1 Categorization and classification: reuse of static structures 365
6.6.2 Poor reusability of some conceptual database models 365
6.6.3 Conceptualizing domain and modeling reality: database models 367
6.6.4 Generic Classes Model 369
6.6.5 Intermediate and Application Classes Models 374
6.7 Unifying real-time and database applications 374

7
Real-time behavioral study beyond UML 381
7.1 SEN or State-Event Net diagram 383
7.1.1 Mathematical foundations of Petri network 385
7.1.2 Using Petri nets in modeling 386
7.1.3 State-Event Net abstractions 387
7.1.4 SEN subnets 389
7.1.5 SEN control structures 389
7.2 UML diagrams mapped into SEN 394
7.2.1 State machine diagram mapped into SEN 394
7.2.2 Activity diagram mapped into SEN 396
7.2.3 Sequence diagram mapped into SEN 397

7.2.4 Exception handling 397
7.2.5 Competition for common resource 399
7.3 Timing constraints with SEN 399
7.4 Case study with SEN 402
7.5 Safety-critical systems 405
7.5.1 State space search with combinatorial method: image attribute methodology 408
7.5.2 Developing safety-critical systems 411
7.5.3 Method of image attributes applied to human and social organizations 414

8
Real time case study 417
8.1 Design of an inclined elevator or track-lift tram 417
8.1.1 Early requirements 417
8.1.2 Requirements engineering: text analysis phase 420
8.1.3 Building classes for instantiating actors in use case diagrams 423
8.1.4 PIM of the inclined elevator system: creating the Generic Classes package 430
8.1.5 PIM of the Inclined Elevator System: creating the Generic Architectures package 434
8.1.6 PIM of the Inclined Elevator System: instantiating a working system 439
8.1.7 PIM of the Inclined Elevator System: behavioral study of the control unit 441
8.2 Emergency service in a hospital and design of a database coupled with a real-time system 448
8.2.1 Early requirements 450
8.2.2 Use case diagrams for System Requirements Specifications Document 452
8.2.3 Designing the patient database coupled with a real-time emergency system 452

References 467
Index 475

List of Figures

2.1.1.1 Odd shapes and free forms 20

2.1.1.2 Objects with/without volume, decorations and adornments 21

2.3.1.1 Gane-Sarson or Yourdon-DeMarco bubble representing a process in functional modeling 24

2.3.2.1 Same Process as Figure 2.3.1.1 with an UML activity diagram 25

2.5.1.1 Typical arrangement of a reactive system 34

2.5.1.2 Thermostatic control built on the principle of a reactive system 35

2.9.0.1 Classic development roadmap 43

2.9.0.2 Instance of development scenario compliant to MDA (to be compared with the classic development roadmap) that enacts high reuse 44

2.9.0.3 Transformations and bridges between models 45

3.2.0.1 Reuse of Common Core Package by UML and MOF 51

3.2.0.2 Four Core packages defined in the Infrastructure 51

3.2.1.1 Core::PrimitiveTypes Package contains four classes 52

3.2.2.1 Contents of Core::Abstractions. This package is composed of 20 elementary packages connected together via a network structure (Fig. 12, Infrastructure Book) 53

3.2.2.2 The Core::Abstractions::Elements Package contains the metaclass Element (Fig. 27 & 28, Infrastructure Book) 53

3.2.2.3 The Core::Abstractions::Ownerships Package expresses the fact that an element can contain other elements (Fig. 50, Infrastructure Book) 54

3.2.2.4 Core::Abstractions::Comments Package (Fig. 20, Infrastructure Book) 55

3.2.2.5 Core::Abstractions::Namespaces Package. A namespace provides a container for named elements (Fig. 48, Infrastructure Book) 56
3.2.2.6 Core::Abstractions::Visibilities Package. This package defines how an element can be visible to other modeling elements (Fig. 62 & 63, Infrastructure Book) 56
3.2.2.7 Core::Abstractions::Expressions Package. Expressions are used to support structured expression trees (Fig. 30, Infrastructure Book) 57
3.2.2.8 The Core::Abstractions::Literals Package specifies metaclasses for specifying literal values. A literal Boolean is a specification of Boolean value. A literal integer contains an integer-valued attribute, etc. (Fig. 40, Infrastructure Book) 58
3.2.2.9 Core::Abstractions::Constraints Package. Constraints use Expressions to define textually conditions that must be satisfied (expressions evaluated to true) in a model (Fig. 23, Infrastructure Book) 58
3.2.2.10 Core::Abstractions::Classifiers Package (Fig. 18, Infrastructure Book) 59
3.2.2.11 Core::Abstractions::Super Package (Fig. 58, Infrastructure Book) 59
3.2.2.12 Core::Abstractions::TypeElements Package (Fig. 61, Infrastructure Book) 59
3.2.2.13 Core::Abstractions::Relationships Package (Fig. 54, Infrastructure Book) 60
3.2.2.14 Core::Abstractions::Generalizations Package (Fig. 32, Infrastructure Book) 61
3.2.2.15 Core::Abstractions::Redeflnitions Package (Fig. 52, Infrastructure Book) 61
3.2.2.16 Core::Abstractions::StructuralFeatures Package (Fig. 56, Infrastructure Book) 62
3.2.2.17 Core::Abstractions::Changeabilities Package (Fig. 16, Infrastructure Book) 62
3.2.2.18 Core::Abstractions::Instances Package (Fig. 35, Infrastructure Book) 62
3.2.2.19 Core::Abstractions::BehavioralFeatures Package (Fig. 14, Infrastructure Book) 63

3.2.2.20 Core::Abstractions::Multiplicities Package (Fig. 42, Infrastructure Book) 63
3.2.2.21 Core::Abstractions::MultiplicityExpressions Package (Fig. 46, Infrastructure Book) 63
3.2.3.1 Core::Basic::Types Package (Fig. 65, Infrastructure Book) 64
3.2.3.2 Core::Basic::Classes Package (reduced version of Fig. 66, Infrastructure Book) 64
3.2.3.3 Data types in the Core::Basic::DataTypes Package (Fig. 67, Infrastructure Book) 65
3.2.3.4 Core::Basic::Packages Package (Fig. 68, Infrastructure Book) 65
3.2.4.1 Core::Constructs::Root Package (Fig. 71, Infrastructure Book) 66
3.2.4.2 Core::Constructs::Classifiers Package (Fig. 84, Infrastructure Book) 67
3.2.4.3 Core::Constructs::Classes Package (Fig. 73, Infrastructure Book) 68
3.2.4.4 Core::Constructs::Operations Package (Fig. 93, Infrastructure Book) 69
3.2.4.5 Core::Constructs::Datatypes Package (Fig. 86, Infrastructure Book) 70
3.2.4.6 Core::Constructs::Packages Package (Fig. 94, Infrastructure Book) 71
3.2.4.7 Core::Constructs::Namespaces Package (Fig. 89, Infrastructure Book) 72
3.2.4.8 Core::Constructs::Expressions Package (Fig. 72, Infrastructure Book) 73
3.2.4.9 Core::Constructs::Constraints Package (Fig. 85, Infrastructure Book) 73
3.2.5.1 Core::Profiles Package (Fig. 109, Infrastructure Book) 74
4.2.0.1 13 diagrams of UML shown as containers 79
4.4.0.1 Top level package dependencies. Relationships are “import” and/or “merge” 88
4.5.0.1 Infrastructure packages merged into the Kernel of UML 88
4.5.0.2 Flat 16 packages of the Kernel merged from the Infrastructure 89

4.5.1.1 Generalization hierarchy of the UML Structure Part showing principal metaclasses and their structure in the metalanguage (Part1) 90

4.5.1.2 Generalization hierarchy of the UML Structure Part (Part2). Operation is a BehavioralFeature and Property is a StructuralFeature 91

4.5.1.3 Generalization hierarchy of the UML Structure Part (Part3) 91

4.6.0.1 Classifier concept of the Kernel package. (Figure 7.9, Superstructure Book) 94

4.6.0 2 Class and Property concepts of the Kernel package. (simplified version of Figure 7.12, Superstructure Book) 95

4.6.1.1 Example of Class Diagram showing a temperature regulation 96

4.6.3.1 Role, multiplicity, navigability, qualifier attached to an association 98

4.6.3.2 E-R diagram replacement 100

4.6.5.1 Using interfaces to describe systems (example 7.57 of Superstructure Book) 104

4.7.0.1 InstanceSpecification concept of the Kernel package (figure 7.14, Superstructure Book) used to instantiate both objects ands links 110

4.7.2.1 Class diagram and Object diagram 112

4.8.0.1 Example of package diagram and relationships defined with packages 115

4.8.0.2 Package concept of the Kernel package (figure 7.14, Superstructure Book) 116

4.9.0.1 StructuredClassifier of the InternalStructures package (figure 9.2, Superstructure Book) 119

4.9.1.1 Differences between a class diagram, a composite structure diagram and an object diagram 121

4.9.2.1 Port metaclass defined in CompositeStructures::Ports package (figure 9.4, Superstructure Book) 122

4.9.3.1 Connector and ConnectorEnd metaclasses as defined in CompositeStructures::InternalStructures package (figure 9.3, Superstructure Book) 123

4.9.4.1 Collaboration and CollaborationUse metaclasses in CompositeStructures::Collaborations package (figure 9.6 & 9.7, Superstructure Book) 124

4.9.4.2 Example in the mechanical domain that illustrates the use of “Collaboration” and “CollaborationUse” modeling concepts and three associated kinds of dependency 125
4.9.4.3 Example elaborated in the house construction domain 126
4.10.0.1 Component concept as defined in BasicComponents package (Fig. 8.2, Superstructure Book) 129
4.10.0.2 Extension of the Component in PackagingComponents package 129
4.11.0.2 Deployment diagram showing components and nodes of a web database 135
4.12.0.1 Generalization hierarchy of the UML Behavior (part1) 136
4.12.0.2 Generalization hierarchy of the UML Behavior (part2) 137
4.12.0.3 Behavior metaclass as defined in BasicBehaviors package 138
4.13.0.1 Hair Dryer State Machine 140
4.13.0.2 Metaclasses StateMachine, State, Region, Transition, Pseudostate, FinalState defined in BehaviorStateMachines package 141
4.13.2.1 Abstractions defined in a state and on a transition 144
4.13.3.1 Trigger and Event metaclasses defines in Communications package 145
4.13.3.2 Subclasses of Event metaclass defined in Communications package 146
4.14.1.1 Activity package dependencies 152
4.14.1.2 Activity metaclass defined in FundamentalActivities (Fig. 12.3 Superstructure, Book) 152
4.14.1.3 Activity metaclass refined in BasicActivities (Fig. 12.5 Superstructure, Book) 153
4.14.1.4 Action metaclass as defined in BasicActions 153
4.14.1.5 Pin metaclass defined in BasicActions package 154
4.14.2.1 Example of activity diagram with activities, decision nodes with conditions, fork and join nodes, initial activity and final activity nodes 156
4.14.2.2 Activity diagrams with partitions and identification of participating objects 157
4.14.3.1 Data flows, action pins and activity parameter nodes 158
4.14.4.1 ObjectNode and ControlNode metaclasses defined in BasicActivities package (Fig. 12.4 Superstructure Book) 160

4.14.4.2 CentralBufferNode defined in JntermediateActivities package and DataStoreNode defined in CompleteActivities package 161
4.14.5.1 Control nodes and their corresponding packages. ActivityFinalNode defined in BasicActivities is enriched in IntermediateActivities 162
4.14.6.1 InterruptibleActivityRegion, ExceptionHandler and ActivityPartition metaclasses 164
4.15.1.1 Main constituents of a sequence diagram: lifelines and messages 170
4.15.2.1 Communication diagram, a replica of the previous sequence diagram 171
4.15.2.2 Partial class diagram of the Car 172
4.15.3.1 Example of Interaction Overview Diagram 173
4.15.4.1 Timing diagram for time specification 175
4.15.5.1 Metaclasses of the Interaction Suite 176
4.16.1.1 Concepts used in a use case diagram 182
4.16.2.1 Concepts used for modeling uses cases (Fig. 16.2, Superstructure Book) 183
4.17.1.1 Example of UML profile 188
5.3.0.1 Mapping of the Business Process Model into UML compatible BP model 203
5.3.1.1 Mapping of the Process Model directly into UML compatible Process model 203
5.3.2.1 Bubble process model of an Order Processing 208
5.3.2.2 Conversion of the DFD "Order Processing" into an activity diagram 210
5.3.3.1 Compatible strategy for adapting an existing methodology to UML 212
5.3.4.1 Overview of Business Process Modeling Notation (BPMN) of an Order Processing in the site of www.bpmi.org 214
5.3.4.2 Order Processing diagram converted from Business Process Modeling Notation (BPMN) to activity diagram notation 215
5.3.6.1 Over simplified version of the Business Organization as inspired from drafts on the OMG site to explain the mapping of business processes 218

5.3.6.2 Reduced version of the Influencer as taken from the OMG site 219
5.3.7.1 Mapping Business Processes in an Enterprise. (See Text) 223
5.3.8.1 Datastores used in the past with DFD. There is no datastore defined in the BPMN and BMM 225
5.3.9.1 Control flows and data flows in a DFD diagram 227
5.3.9.2 Conversion of the DFD of figure 5.3.8.1 into activity diagram 230
5.4.2.1 States of the Hair Dryer not decomposed. 64 states are possible at the beginning, based on 6 binary operations (activated or not) 235
5.4.2.2 States of the Hair Dryer decomposed into two separate objects. The number of states is 16 (8+8) 235
5.4.2.3 Conventional state used with UML is not elementary state (figure 15.32 Superstructure book) 237
5.4.8.1 Transition characteristics in state machine diagram. Transitions are made of known concepts exposed in previous chapters 254
5.5.1.1 Car class declared in three different contexts 265
5.5.3.1 Relationship between Relationship, Association, Link, Connector and Dependency 271
5.5.4.1 Attribute of relation in ER diagram 272
5.5.4.2 Association class. When a person takes a job in a company, the two sides are tied by a contract that is represented as a class association 272
5.5.4.3 Association Class. Association class represents a contract in business domain 273
5.5.4.4 The first example shows a misuse of 4-ary relationship, corrected in the first step to a 3-ary relationship, then corrected in a second step to transform the ternary association into two binary associations (see text for discussion) 276
5.5.5.1 Same objects but two different aggregation criteria that are “Administrative Structure” and “Geographical Localization” 278
5.5.5.2 Decomposition that respects the level semantic coherence 279
5.5.6.1 Example of inheritance hierarchy 280

5.5.7.1 In object technology, multiple inheritance use union of attributes and operations (not intersection), so the child class is essentially the union of the behavior of the two parent classes 282

5.5.7.2 Classical problem known as the "diamond" problem [Snyder, 1986] in multiple inheritance 283

5.5.7.3 Inheritance hierarchy containing simple and multiple inheritance 284

5.5.7.4 Example of questionable multiple inheritance hierarchy 285

5.5.7.5 Composition or multiple inheritance? The first solution is a multiple inheritance with some minor conflict resolution and the second is a team of three engineers 286

5.5.8.1 Representation of role in UML as name of an association end 287

5.5.8.2 Roles are modeled as separate classes 287

6.1.1.1 Object diagram of a Home Alarm System showing the modeling of visible and invisible, human and non human objects 293

6.1.2.1 Sequence and class diagrams showing the arrangement of a parking ticket delivery system from the user viewpoint 295

6.1.3.1 At the starting point, we have 4 independent pieces Mechanical_Piece_1, Mechanical_Piece_2, Bolt and Nut At the end, we have a monolithic piece shown in the figure 297

6.1.3.2 Translation of mechanical assembly into objects and messages 298

6.1.4.1 Variable speed electric fan with its power supply 299

6.1.4.2 Object diagram representing a control model 299

6.1.4.3 Use of a Monitor object 302

6.2.3.1 UML activities diagram showing activities of Requirements Engineering phase 309

6.3.2.1 Class diagram of the RMSHC system 318

6.3.2.2 Use case diagram depicting goals enumerated for the RMSHC 321

6.3.2.3 Activities of the Blood Pressure Monitor 323

6.3 2.4 Use case diagram showing activities related to the management of the Electronic Pifi Box 323

6.3.3.1 Package diagram of the Requirements Analysis of the Remote Monitoring System for Health Conditions 324

6.3.3.2 Actors are connected directly through inheritance only 324
6.3.3.3 When actors are connected together, we have an object style connection. In a use case diagram, a use case is inserted between two actors meaning that actors are involved in the activity 325
6.3.3.4 Inheritance between roles. The left diagram is replaced by the right diagram 325
6.3.4.1 Business Process Diagram showing the functional aspects of the Pill Management Software Component of the RSMHC project 327
6.4.1.1 A link Model traces all mappings between the SRS elements and components of PIM set and PSM set 330
6.4.3.1 Design process roadmap represented by a business process diagram combined with an object diagram 332
6.5.1.1 Decomposition of a Security System in an Airport 338
6.5.1.2 Re-specs with use cases and packages illustrated via the RMSHC example 339
6.5.3.1 Example of call graph showing all execution paths in a program 341
6.5.3.2 User's Instruction "How to arm a home alarm system" expressed by a sequence diagram 342
6.5.3.3 Corresponding classes identified while drawing the sequence diagram 343
6.5.3.4 Sequence diagram for developing the Home Alarm System 344
6.5.3.5 The status of a passive component in a real time system can be detected by reading the status of the component with a busy wait loop or mounting appropriate hardnre to send an interrupting signal to the Controller 346
6.5.5.1 Interaction Overview Diagram packing many references to sequence diagrams inside an activity diagram 348
6.5.5.2 Activity diagram as a starting method for the design 349
6.5.5.3 State machine diagram as a starting method for designing 350
6.5.6.1 Domain engineering with two activity flows (Domain Engineering & Client System Development) that may be conducted in parallel or not 352
6.5.6.2 Classes of Alarm System Domain (Generic Classes Package) 353
6.5.6.3 Contents of Generic Architectures Package (Alarm System) 355
6.5.6.4 Contents of Reusable Components Package (Alarm System) 356

6.5.6.5 Instance of Customer application (Alarm System). A Customer application contains objects instead of classes 358

6.5.7.1 State machine diagram of the Panel Controller showing activities accompanying the arming process 361

6.6.0.1 Physical, logical, conceptual and semantic models on a user perception axis of low or high level of data representation 364

6.6.2.1 Classic conceptual Person class often seen in database model 366

6.6.2.2 Slight correction of the model of figure 6.6.2.1 366

6.6.3.1 Organization of data models, From Generic Classes Model to the Physical Models 368

6.6.4.1 Elementary GCM elements of Humans, Identifications, Locations, Immobilizations packages illustrating the contents of Generic Classes Model 370

6.7.0.1 Deployment diagram explaining that many bagRT are created temporarily (they can be made persistent locally) 378

7.0.0.1 The Dynamic Model developed with SEN (State Event Net) acts as intermediate model between PSM and Code 383

7.1.1.1 Petri net with 4 places P1-P4 and 2 transitions T1-T2 386

7.1.2.1 Modeling task execution with a Petri net 387

7.1.3.1 Various graphic representations of SEN places 389

7.1.4.1 SEN subnets 390

7.1.5.1 Various End of Activity representation and Interruptible Activity 391

7.1.5.2 Decision nodes (a) simple IF, (b) CASE with one outgoing edge, (c) handling complex case with overlapping conditions and possibility of parallel split 392

7.1.5.3 Example of DO loop represented by a SEN at two levels, first at programming language level and at design level (as a sub SEN) 393

7.1.5.4 Fork, Join and Merge nodes 394

7.2.1.1 Mapping of a state diagram into SEN 395

7.2.2.1 Conversion of activity diagram with object flow into SEN 396

7.3.2.1 Intra-object interactions and inter-object interactions 398

7.2.4.1 Exception handling in a SEN 399

7.2.5.1 Competition for a common resource 400

7.3.0.1 Representation on the End of Activity (EOA) and Before End of Activity (BEOA) conditions 401
7.3.0.2 Timing specification with variable "end" 401
7.3.0.3 Timing dependence between two objects A and B with EOA and BEOA 402
7.4.0.1 Use case diagram representing activities of the Tamagotchi, Person and Timer 404
7.4.0.2 SEN representing the state evolution of the Tamagotchi 404
7.4.0.3 Class and object diagrams of the Tamagotchi 405
7.4.0.4 SEN versions of the Tamagotchi application with separate SEN. Cross-interactions between objects are not shown but they are parts of class and object diagrams 406
7.5.1.1 Attributes and image attributes of an object 409
7.5.1.2 SEN of the Tamagotchi established after an extensive study with the method of image attributes 412
8.1.1.1 Free form diagram showing main components of the inclined elevator 419
8.1.1.2 Control Panel of the inclined elevator 419
8.1.3.1 Classes of the Inclined Elevator identified by Text Analysis of Early Requirements Specification 424
8.1.3.1 Thirteen use case diagrams showing activities of the Inclined Elevator System from which CRT parameters must be evaluated 430
8.1.4.1 Generic Classes Package in use in LifiTram Company 431
8.1.4.2 Details of the Generic Mechanical Classes Package. These classes are specific to the Elevator Domain 431
8.1.4.3 Details of the Generic Electromechanical Classes Package 432
8.1.4.4 Details of the Generic Sensor Classes Package 433
8.1.4.5 Details of the General Inteffigent Devices Classes Package 434
8.1.5.1 For one shot deal project with small chance of recurrence (small scale reuse), components can be built directly from objects instantiated from the Generic Classes package. This diagram is incomplete and is targeted to show small-scale reuse process 435
8.1.5.2 Four components are defined in Generic Architectures Package 436

8.1.5.3	Generic Architectures Package contains the definition of four composite components	436
8.1.5.4	Composition of the Cable-Transmission-Tension component	437
8.1.5.5	Composition of the Track-Limit–Sensors component	437
8.1.5.6	Composition of the Cabin-Door-Detector-Castor-Overspeed-Obstruction-TrackBrake component	438
8.1.5.7	Composition of the Control Unit component	439
8.1.6.1	The Package Application Objects/Components contains the PIM built from merging classes and components from other packages. Only Application Classes Package is not reusable	440
8.1.6.2	The final system is composed of objects/components instantiated from reusable Generic Classes Package, reusable Generic Architecture Package	440
8.1.7.1	Object diagram of the Inclined Elevator Controller with all inputs/outputs of its first neighbors	444
8.1.7.2	When the power is on, the Controller undergoes a system check routine then powers the VFD. If faults are declared, the Controller requests that all faults must be cleared before enabling any movement	445
8.1.7.3	This routine details "System Check & Power on VFD" subroutine. It sets the main EM brake on, verifies track jam, slack cable and overspeed sensors and set lights accordingly	446
8.1.7.4	The subroutine "Update Fault Indicators" reads all sensors and buttons then updates all signal lights (Lights are exclusive, Red or Green only)	447
8.1.7.5	The main subroutine supports the normal operation mode of the cabin	449
8.2.2.1	Classes of the Hospital with the Emergency Classification Problem. They are used to instantiate objects for specification purpose only, not for design	453
8.2.2.2	UC0 to UC8 specifying the hospital emergency problem	457
8.2.3.1	Contents of the Generic Classes Model (GCM). Figures of this GCM suite must be completed with Figure 6.6.4.1 of chapter 6	459

8.2.3.2 Contents of the ICM package that contains classes belonging to the Health Care System 462
8.2.3.3 Contents of the ACM package (Application Classes Model). This package relates to all relationships specific to the Hospital X 464

List of Tables

1.1.1.1 The table shows the similarity of approaches and steps used to design a system or solve a problem 4

1.2.1.1 Characteristics of real time applications 11

1.3.1.1 Depletion of vocabulary from high to lower models 14

2.8.0.1 Survey questions to test the commitment to reuse concept in an organization 42

3.1.0.1 4-layer architecture of UML 48

4.2.0.1 Six structural diagrams of UML standard 81

4.2.0.2 Seven behavioral diagrams of UML 2 standard. Behavioral includes both functional and dynamic aspects 82

4.3.1.1 Parallel between the natural language and UML 84

4.4.0.1 Four compliance levels of UML L0 to L3 defined with regard to the abstract syntax dimension 87

4.6.7.1 Modeling concepts mostly used in a class diagram 105

4.6.8.1 Standard dependencies defined in UML. Users can freely define their proper set of dependency stereotypes 108

4.7.3.1 Modeling concepts found in an object diagram 113

4.8.1.1 Modeling concepts found in a package diagram. — 117

4.9.5.1 Modeling concepts in a composite structure diagram 126

4.10.1.1 Modeling concepts of a component diagram 130

4.11.0.1 Modeling concepts used in a deployment diagram 132

4.13.5.1 Modeling concepts used in a State Machine diagram 147

4.14.1.1 Seven levels of Activities definitions 155

4.14.7.1 Modeling concepts used in an Activity diagram 164

4.15.6.1 Modeling concepts of the Interaction Suite 176

4.16.3.1 Concepts used in a use case diagram 184

4.17.2.1 Modeling concepts used in profiles 188
5.1.1.1 Mapping of Abstraction concept and different meanings in the domain of modeling 192
5.1.2.1 Mapping of a simple concept that necessitates just a categorization process with a Class 194
5.1.2.2 Mapping of a "Complex Concept" concept and example 195
5.1.3.1 Mapping of a Domain concept in UML 19
5.1.4.1 Mapping of an Ontology concept in UML 198
5.1.5.1 Mapping of a "model" concept into UML 199
5.2.0.1 Vocabulary depletion when evolving from application domains towards programming domain 202
5.3.2.1 Mapping rules of a simple Process Model into UML 209
5.3.7.1 Mapping of process into UML metaclasses 221
5.3.8.1 Mapping of a datastore into UML concepts 226
5.3.9.1 Meaning of UML control flow, object flow, interrupt edge 229
5.4.1.1 Description of UML concepts related to the real world concept of "state" 232
5.4.4.1 Support for concept of event in UML 242
5.4.5.1 This table explains how UML deals with conditions at various levels 244
5.4.6.1 Support of Message concepts in UML 249
5.4.8.1 Mapping of system change, system evolution, system transition or system movement into UML transition and UML control flow 254
5.5.4.1 Table containing all possible entries of a ternary association between Participant, TradeShow and Location 275
5.5.5.1 Aggregation is considered as the loosely coupled form and composition the highly coupled form between parts and whole 277
6.2.3.1 Sub phases of the Requirements Engineering process 312
6.3.1.1 Convention used in this book: basic concepts in use case diagram and extensions of concept interpretation 314
6.3.2.1 Example of Text Content Analysis in order to establish use case diagrams structuring requirements 319
7.4.0.1 Text Content Analysis of the Tamagotchi specification 403
7.5.1.1 Table of attributes and image attributes of the Tamagotchi 411

7.5.1.2 Table of Sequences built from variations of state values of Tamagotchi regular attributes and image attributes 413
8.1.2.1 Text analysis of Early Requirements (Inclined Elevator). First step of Requirements Engineering phase 420
8.1.7.1 This table lists all objects/components constituting the first neighbors of the Controller and identifies the nature of communication channels exchanged with the Controller 442

Preface

Book Description

Real-Time Object Uniform Design Methodology with UML is a theoretical and practical book written for busy people who want to untangle the complex world of system development, find essential materials without digging in UML standard documentation, grasp subtle concepts of object orientation, practice the new Model Driven Architecture (MDA), experience the reuse mechanism, and transform the bare metal programming of real-time and embedded products into more handsome platform-independent and platform-specific components. With this rapid methodology of development, practitioners can spare time, avoid tons of written documentation by relieving this tedious task to smart CASE (computer-aided software engineering) tools, and have a quick and synthetic view of any system through a well-built set of pictures and blueprints.

The methodology presented in this book is a *neutral methodology* based on a thorough study of fundamental modeling concepts and then a temporary mapping of these concepts on current available standards and tools. We say "temporary" because research is in fact a never-ending activity. Good standards are evolving standards and the truth is always questionable. We are not pretending to add a new methodology to the numerous existent or in-house methodologies. We hope that the reader is able to catch the thoughts presented in this book to have a more critical view on any future methodology (a kind of meta "methodology"). So, feel free to prune off parts that you do not feel comfortable with. On the other hand, give us also the opportunity to correct any inaccuracy or vagueness in future versions of this book to always keep it at the leading edge of knowledge.

Why do we need object technology and an innovative uniform view of the message concept? Neil Armstrong had already walked on the Moon in 1969 before the maturity of object technology. Most real-time methodologies in the past were good guidelines at that time. Today, we know that every system has exactly three views: structural, functional, and dynamic ("behavioral" combines "functional" and "dynamic"). We must know all about systems before building them. At the implementation phase, algorithms are inferred from dynamic analyses. Before serious dynamic investigations, we must first structure the systems. In the past, real-time practitioners tackled systems through functions, along the functional axis. Database developers unveiled the first structural aspects with entities and relationships, and so approached problems from the structural axis. Mixed and multidisciplinary software, hardware, physical, biological, and social systems suffered from the use of different tools to explore and design multidomain systems. Object methodologists, who made the proposal of class abstraction, which contains functions, structures, and dynamic contents, claimed that every software system can be modeled with object paradigm and blueprinted with a tool like the UML. Effectively, we observe that object technology can be used to study any complex multidisciplinary system if we supply a more uniform view of the message concept to represent interaction between objects of very different nature. We therefore propose some "patches" in object paradigm, based on "induced energy," "cause–effect chain," and the generalization of the "message concept." They open unsuspected doors for the modeling of a richer set of multidomain applications.

Is object technology compatible with business processes? If everyone is convinced that object technology is the right way to go, in university school benches, why do students still make classes from each function they discover even in the presence of an object environment like Java, Ada, C++, or C# and UML modeling tools? Why do people get stuck with process? In fact, even with object technology, a process that is a real working entity in an object system still plays exactly the same role as it played in the past; object technology only dictates how to build and organize them smartly to satisfy other software engineering goals and deal with complexity. The functional view of systems remains at the foreground contrary to some prognostics of object theorists of the first wave. Processes become more structured and find their strength mainly in business models. After an exposé on business process model, we connect business models to the UML and give an answer to the question as to whether a process could be mapped into a full class or not.

Why MDA? Large-scale and multidisciplinary systems need complementary techniques. A biometric measurement system is, for instance, a multidisciplinary embedded system. An airport management is a large-scale mixing of databases, humans, and real-time components. It is well known that a large project life cycle that needs an analysis, design, implementation, and validating phases must be subdivided into sub- and sub-subprojects. This hierarchical view of the development model is not sufficient and should be revised and adapted to the convergence point of the black art of design and the recent MDA concept. The MDA deals with "models" and begins with a *Platform Independent Model* (PIM) that represents the system in terms understood by model practitioners who are often not computer specialists or implementers. An object-oriented and logical architecture is then built to represent the planned solution. Communication between domain experts and system architects is conducted to validate the PIM that can be simulated and tested. *Platform Specific Model* or PSM signs up as the intermediate phase before coding or implementing. Models facilitate communication, reuse, and structure the development process instead of the product itself. This book has clearly a convergence point with the MDA proposal.

What is the double problem of a system designer? When we develop a system, we are facing a double problem: building the right system, deploying appropriate human and technological resources to realize this system in a reasonable time frame and at an acceptable cost. In an organization business model, the managerial structure should be geared to the development process that involves a plethora of recurrent problems including management of the development process, looking for the right individuals, finding available resources, etc. Development of a product or a design of a solution for a complex system is not simple at all. The batch of stress, bad luck, or mitigated success, at personal or organizational levels, are clues showing the complexity of our daily business. Things may appear "simple" because we have experience, we react by reflex, and we apply consciously or unconsciously known solutions or resolution patterns that transform all complex processes into automated daily acts or reactions. As problems evolve slightly with time and context, systematical patterns, unified way of doing things replicated in "similar" situations, based on the erroneous thinking that they do save cost, may propose approximate solution or inadequate systems, blur nuances, and finally may create more problems in the longer term than they can solve. Moreover, patterns or ready-to-use solutions targeted to have rapid results, if applied without any methodology, shortcuts inevitably the analysis and design processes, and may impair the activity of finding the optimized solution or system. Another problem is unexpected constraints that may be added to the requirement analysis and alter the original problem. For instance, in political systems, under the pressure of the media, some hasty decisions based on commonly admitted patterns, taken under the pressure of the population in catastrophic circumstances, can calm down the media pressure but may represent atrocious examples of resource wasting. From a perspective of system and solution design, we are first solving the media problem by applying a pattern to the original problem quickly without any detailed analysis of the main problem.

Facing the MDA and current research trend, we can say that a methodology is not a mechanical act of applying patterns shortcutting analysis and design phase, but it proposes a way of quickly creating appropriate patterns for each new problem or system, reusing models. A methodology does not give up the notion of patterns or existing generic models. By organizing the development process into a multitude of models, we can easily distill universal patterns applicable to several domains, those that are specific to a particular domain and finally those that must be generated on the fly for a given situation. Experience does not mean ready-to-use solutions and patterns shortcutting analysis and design phases (they simplify them considerably instead) but suggests a process of reusing both universal and specific domain knowledge to design an optimized solution taking into account requirements of the current context. Parts of the solution patterns can be returned to the universal or domain knowledge to enrich the overall experience that constitutes the business assets of an organization or individual. This process is the essence of the reuse concept exposed in this work, and constitutes the immediate benefit of object technology and the MDA.

Audience

Designed for analysts, object programmers, system architects, application developers, project managers, and researchers, this book contributes an effort to implement multidomain and complex systems mixing databases, real-time controls, humans, etc. with object technologies. Large-scale applications are handled with complexity reduction techniques so, at any stage of system development, individual projects are self-contained and easily manageable. It is very difficult to find a complete and well-managed object project in the literature. Undoubtedly, they do exist and belong to well-kept industrial secrets. Well-built object software is still the black art of few specialists. Could this bunch become more crowded in the future?

This book can also be used as a graduate-level text for students at any domains involved in modern software development to inoculate intelligence to their biological, mechanical, electrical, physical, and social systems. They can discover that the UML and the MDA can be used in their fields unexpectedly. Specifically for computer science and software engineering domains, this book can be taught starting at the 4th trimester of undergraduate level when students have enough background on programming (an object programming course and a relational database course are necessary requisites). The reader should be exposed to difficult problem-solving situations and confronted with a large-scale project to really appreciate this neutral and uniform methodology.

Book Organization

From the scientific literature production domain, *this book is designed with our best knowledge about learning object research domain*, so individual knowledge chunks are often self-contained and self-documenting. If their knowledge dependency hierarchy and matrix cannot be avoided, superfluous cross-references/deferments are pruned to minimize learning effort.

Chapter 1 introduces modern software/hardware, database/real-time control, computer/intelligent, hybrid systems, and multidisciplinary systems. Chapter 2 explains techniques to cope with the complexity of large-scale systems. Chapter 3 introduces the metalanguage of the UML as a lesson of rigorous and metasystem development. This helps developers to acquire a deep understanding of metaelements of the UML. This chapter can be skipped at first reading. Chapter 4 is the main chapter for learning how to use the 13 diagrams of the UML; it examines the nature of each UML concept. One or many short examples accompany each theoretical exposé. Chapter 5 exposes fundamental concepts of the real world and their mappings into UML modeling elements. Their semantics are taken from problem solving and system development viewpoint, independently from the way the UML considers them. Business process diagrams in business domains are interesting examples for mapping UML diagrams. Chapter 6 presents the uniform methodology targeted to study complex multidisciplinary systems. A uniform view of the message concept is needed to represent interactions between objects of different nature. Some "patches" in object paradigm, based on "induced energy," "cause–effect chain," and the generalization of the "message concept" are introduced in order to model a richer set of applications. In Chapter 7, two advanced research topics beyond the UML are exposed to complement the arsenal of development tools. First, a state-event network (SEN), a new diagram based on Petri net, supports dynamical studies and refines the UML behavioral diagrams before implementation. Second, the "image attribute method" is exposed to systematically build sequences of a real-time system. All sequences melted together constitute the algorithm of the system or the solution. This method is a combinatorial technique deployed to study dynamic behavior of safety-critical systems. Chapter 8 contains case studies, both in technical and social domains.

Informative, actual, useful, returns from intellectual investment are targeted along this book-designing process.

Acknowledgments

All diagrams in this book are drawn with the version 6.5 of Enterprise Architect implementing UML 2.1, graciously donated by Sparx Systems Inc. SEN diagrams are drawn with Visio 2003. We would like to thank Rational, IBM, and Sparx Systems for supporting computer science and software engineering courses at Laval University, Quebec, Canada, with their software products year after year. We would like to thank OMG (Object Management Group) organization for providing the scientific community with invaluable object standards, for being the opposite of a conservative organization, and for graciously making numerous documentations available on their site. This work could not have been completed without the constant support of my wife Moc Mien, friends and relatives who were always there when positive morale, food, and scientific supports must be combined to produce words. Finally, a special thanks to the Chief Editor Mark De Jongh, and Cindy Zitter for accompanying us at every step along this arduous project.